

# Schedulability Analysis of the Linux Push and Pull Scheduler with Arbitrary Processor Affinities

Arpan Gujarati   Felipe Cerqueira   Björn B. Brandenburg  
*Max Planck Institute for Software Systems (MPI-SWS)*

**Abstract**—Contemporary multiprocessor real-time operating systems, such as VxWorks, LynxOS, QNX, and real-time variants of Linux, allow a process to have an arbitrary processor affinity, that is, a process may be pinned to an arbitrary subset of the processors in the system. Placing such a hard constraint on process migrations can help to improve cache performance of specific multi-threaded applications, achieve isolation among components, and aid in load-balancing. However, to date, the lack of schedulability analysis for such systems prevents the use of arbitrary processor affinities in predictable hard real-time applications. In this paper, it is shown that job-level fixed-priority scheduling with arbitrary processor affinities is strictly more general than global, clustered, and partitioned job-level fixed-priority scheduling. The Linux push and pull scheduler is studied as a reference implementation and techniques for the schedulability analysis of hard real-time tasks with arbitrary processor affinity masks are presented. The proposed tests work by reducing the scheduling problem to “global-like” sub-problems to which existing global schedulability tests can be applied. Schedulability experiments show the proposed techniques to be effective.

## I. INTRODUCTION

As multicore systems have become the standard computing platform in many domains, the question of how to efficiently exploit the available hardware parallelism for real-time workloads has gained importance. In particular, the problem of scheduling multiprocessor real-time systems has received considerable attention over the past decade and various scheduling algorithms have been proposed.

One of the main dimensions along which these real-time scheduling algorithms for multiprocessors are classified is the permitted degree of migration. *Global* and *partitioned* scheduling represent two extremes of this spectrum. Under global scheduling, the scheduler dynamically dispatches ready tasks to different processors from a single queue in the order of their priorities, whereas under partitioned scheduling, each task is statically assigned to a single processor, and each processor is then scheduled independently. Researchers have also studied hybrid approaches in detail. One notable hybrid approach is *clustered scheduling*, under which processors are grouped into disjoint clusters, each task is statically assigned to a single cluster, and “global” scheduling is applied within each cluster.

Interestingly, many contemporary real-time operating systems, such as VxWorks, LynxOS, QNX, and other real-time variants of Linux, do not actually implement the schedulers as described in the literature. Instead, they use the concept of *processor affinity* to implement a more flexible migration strategy. For example, Linux provides the system call `sched_setaffinity()`, which allows the processor

affinity of a process or a thread to be specified, with the interpretation that the process (or the thread) may not execute on any processor that is not part of its processor affinity. That is, processor affinities allow binding<sup>1</sup> a process to an arbitrary subset of processors in the system.

Processor affinities are commonly used in throughput-oriented computing to boost an application’s performance (e.g., TCP throughput on multiprocessor servers [21]) and to completely isolate real-time applications from non-real-time applications by assigning them to different cores. Processor affinities can also be used to realize global, partitioned, and clustered scheduling. (E.g., to realize partitioned scheduling, each task’s processor affinity is set to include exactly one processor, and to realize global scheduling, each task’s processor affinity is set to include all processors.) However, what makes this feature interesting from a scheduling point of view is that *arbitrary processor affinities* (APAs) can be assigned on a task-by-task basis, which permits the specification of migration strategies that are more flexible and less regular than those studied in the literature to date.

In this paper, we present the first techniques for the schedulability analysis of sporadic tasks with APAs under *job-level fixed-priority* (JLFP) scheduling, both to enable the use of APAs in *predictable* hard real-time systems (i.e., to allow APAs to be used in systems in which the timing correctness must be established *a priori*), and to explore whether scheduling with APAs (*APA scheduling* hereafter) merits increased attention from the real-time community.

In particular, we answer three fundamental questions pertaining to APA scheduling. **(i)** Is APA scheduling strictly dominant w.r.t. global, partitioned, and clustered scheduling? (Yes, for JLFP policies, see Sec. III.) **(ii)** Is it possible to analyze APA scheduling using existing techniques for global schedulability analysis? (Yes, see Sec. IV.) **(iii)** Besides the desirable ability to isolate tasks for scheduling-unrelated reasons such as security concerns, does APA scheduling also offer improved schedulability? (To some extent, yes, see Sec. V.)

To answer these questions, we study Linux’s push and pull scheduler as a reference implementation of APA scheduling and derive matching schedulability analysis.

### A. Prior Work and Background

Recall that APA scheduling uses the concept of processor affinities to implement a flexible migration strategy. Therefore,

<sup>1</sup>In the context of migrations, *binding* means that a process can only migrate to (or be scheduled on) the processors that it is bound to.

we start by classifying real-time scheduling algorithms according to different migration strategies and compare them with APA scheduling. We then classify different priority assignment policies used in real-time scheduling and discuss how they relate to APA scheduling. Finally, we compare APA scheduling to scheduling problems of a similar structure in related domains.

According to the degree of migrations allowed, real-time scheduling algorithms either allow *unrestricted* migrations, *no* migrations, or follow a *hybrid* approach with an intermediate degree of migration. Global scheduling [15] is an example of scheduling algorithm that allows unrestricted migration of tasks across all processors (if required) while partitioned scheduling [15] is an example of a scheduling algorithm that allows no migration at all. Some notable hybrid scheduling algorithms that have been proposed include the aforementioned clustered scheduling [6, 14], *semi-partitioned* scheduling (e.g., see [1, 5, 13, 25]) and *restricted-migration* scheduling (e.g., see [1, 18]).

APA scheduling *generalizes* global, partitioned, and clustered scheduling. In other words, APA scheduling constrains each task to migrate only among a limited set of processors defined by the task’s processor affinity. Therefore, using appropriate processor affinity assignment, a task set can be modeled as a global, clustered, or partitioned task set (see Sec. III).

Under semi-partitioned scheduling, most tasks are statically assigned to one processor (as under partitioning) and only a few tasks migrate (as under global scheduling). APA scheduling resembles semi-partitioned scheduling in that it may also allow two tasks to have separate degrees of migration. However, if and when a task migrates under APA scheduling is determined dynamically “on-demand” as under global scheduling, whereas semi-partitioned schedulers typically restrict tasks to migrate at pre-determined points in time to pre-determined processors.

APA scheduling, which restricts migrations to occur among a fixed set of processors, should also not be confused with restricted-migration scheduling. Under restricted-migration scheduling, migrations occur only at job boundaries. It limits *when* a job may migrate, whereas APA scheduling (like global, clustered, and semi-partitioned scheduling) primarily specifies *where* a job may migrate to. However, both global and semi-partitioned scheduling [1, 18] can be combined with restricted-migration scheduling and a similar approach could also be taken for APA scheduling.

Orthogonal to the degree of migration allowed, scheduling algorithms also have a choice of how to prioritize different jobs or tasks in a task set and how these priorities may vary over time. In particular, the different priority assignment policies used in real-time scheduling can be classified either as *task-level fixed priority* (FP), *job-level fixed priority* (JLFP), or *job-level dynamic priority* (JLDP) policies.

A FP policy assigns a unique priority to each task, e.g., the classic *Rate Monotonic* (RM) [29] and *Deadline Monotonic* (DM) [4, 28] priority assignments fall into this category. A JLFP policy assigns a fixed priority to each job, and unlike under FP policies, two jobs of the same task may have distinct priorities; e.g., this is the case in the *Earliest Deadline First* [29]

policy. A JLDP policy allows a job to have distinct priorities during its lifetime; a prominent example in this category is the *Least Laxity First* [17] policy. APA scheduling can be combined with any of these priority assignment policies.

*In this paper, we restrict our focus to JLFP policies, since such policies can be implemented with low overheads [12].* However, our results pertaining to JLFP policies also apply to FP scheduling, which is the policy actually implemented in mainline Linux.<sup>2</sup> Similar to priorities under a FP policy, we assume that all jobs of a task share the same processor affinity (i.e., the processor affinity assignment does not vary over time).

APA scheduling could also be understood as global scheduling on a (degenerate) *unrelated heterogeneous multiprocessor* (e.g., see [22]), where each task has the same, constant execution cost on any processor included in its processor affinity, and “infinite” execution cost on any other processor. However, such platforms have primarily been studied in the context of partitioned scheduling to date (e.g., see [2, 8, 22]).

Finally, the APA scheduling problem also resembles a classic non-real-time scheduling problem in which a set of non-recurrent jobs is to be scheduled on a set of *restricted identical machines* [23, 27], i.e., given a set of  $n$  jobs and a set of  $m$  parallel machines, where each job has a processing time and a set of machines to which it can be assigned, the goal is to find a schedule that optimizes a given objective (e.g., a schedule with a minimal *makespan*). However, to the best of our knowledge, this problem has not been studied in the context of the classic sporadic task model of recurrent real-time execution (or w.r.t. other recurrent task models).

## B. System Model

We consider the problem of scheduling a set of  $n$  real-time tasks  $\tau = \{T_1, \dots, T_n\}$  on a set of  $m$  identical processors  $\pi = \{\Pi_1, \Pi_2, \dots, \Pi_m\}$ . We adopt the classic *sporadic task model* [30], where each task  $T_i = (e_i, d_i, p_i)$  is characterized by a *worst-case execution time*  $e_i$ , a *relative deadline*  $d_i$ , and a *minimum inter-arrival time* or *period*  $p_i$ . Based on the relation between its relative deadline and its period, a task  $T_i$  either has an *implicit* deadline ( $d_i = p_i$ ), a *constrained* deadline ( $d_i \leq p_i$ ), or an *arbitrary* deadline. The utilization  $u_i$  of a task  $T_i$  is  $e_i/p_i$  and the density  $\delta_i$  of a task  $T_i$  is  $e_i/\min(d_i, p_i)$ .

Each task  $T_i$  also has an associated processor affinity  $\alpha_i$ , where  $\alpha_i \subseteq \pi$  is the set of processors on which  $T_i$  can be scheduled. In this initial work on the analysis of APA scheduling, we assume that  $\alpha_i$  is static, i.e., processor affinities do not change over time. We define the joint processor affinity  $cpus(\gamma)$  of a task set  $\gamma$  as the set of processors on which at least one task in  $\gamma$  can be scheduled. Similarly, for a set of processors  $\rho$ ,  $tasks(\rho)$  defines the set of tasks that can be scheduled on at least one processor in  $\rho$ .

$$cpus(\gamma) = \bigcup_{\forall T_i \in \gamma} \alpha_i, \quad tasks(\rho) = \{T_i \mid \alpha_i \cap \rho \neq \emptyset\} \quad (1)$$

<sup>2</sup>Since first publication of the paper, an EDF implementation has been added to mainline Linux as well [26].

A task  $T_k$  can (directly) *interfere* with another task  $T_i$ , i.e., delay  $T_i$ 's execution, only if  $\alpha_k$  overlaps with  $\alpha_i$ . We let  $I_i$  denote the set of all such tasks in  $\tau$  whose processor affinities overlap with  $\alpha_i$ . In general, the exact interfering task set depends on the scheduling policy. Therefore, we define  $I_i^A$  as the interfering task set if  $T_i$  is scheduled under scheduling algorithm  $A$ . For example, in an FP scheduler, only higher-priority tasks can interfere with  $T_i$ . If we let  $prio(T_k)$  denote  $T_k$ 's priority, where  $prio(T_k) > prio(T_i)$  implies that  $T_k$  has a higher priority than  $T_i$  (i.e.,  $T_k$  can preempt  $T_i$ ), then

$$I_i^{FP} = \{T_k \mid prio(T_k) > prio(T_i) \wedge \alpha_k \cap \alpha_i \neq \phi\}. \quad (2)$$

For simplicity, we assume integral time throughout the paper. Therefore, any time instant  $t$  is assumed to be a non-negative integral value representing the entire interval  $[t, t + 1)$ . We assume that tasks do not share resources (besides processors) and do not suspend themselves, i.e., a job is delayed only if other tasks interfere with it. Further, a task  $T_i$  is *backlogged* if a job of  $T_i$  is available for execution, but  $T_i$  is not scheduled on any processor. We also use two concepts frequently: *schedulability of a task* and *schedulability of a task set*. A task  $T_i \in \tau$  is schedulable on the processor platform  $\pi$ , if it can be shown that no job of  $T_i$  ever misses its deadline. A task set  $\tau$  is schedulable on the processor platform  $\pi$  if all tasks in  $\tau$  are schedulable on  $\pi$ .

### C. Paper Organization

The rest of this paper is structured as follows. In Sec. II we give a brief overview of the Linux push and pull scheduler. We also give a formal definition of an APA scheduler, assuming the Linux scheduler as a reference implementation of APA scheduling. In Sec. III, we compare APA scheduling with global, partitioned, and clustered scheduling from a schedulability perspective. In Secs. IV and V, we present schedulability analysis techniques for APA scheduling and evaluate them using schedulability experiments. Lastly, Sec. VI discusses future work and concluding remarks.

## II. PUSH AND PULL SCHEDULING IN LINUX

The Linux kernel employs an efficient scheduling framework based on processor-local queues. This framework resembles the design of a partitioned scheduler, i.e., every processor has a runqueue containing backlogged tasks and every task in the system belongs to one, and just one, runqueue. Implementing partitioned scheduling is trivial in this design by enforcing a no-migration policy (i.e., by assigning singleton processor affinities). However, the Linux scheduler is also capable of emulating global and APA scheduling using appropriate processor affinities and migrations. We review the Linux scheduler implementation of global and APA scheduling in the remainder of this section to illustrate the similarities between these two approaches, which we later exploit to derive schedulability analysis techniques for APA scheduling.

### A. Global Scheduling with Push and Pull Operations

Under global scheduling, all backlogged tasks are conceptually stored in a single priority-ordered queue that is served by

all processors, and the highest-priority tasks from this queue are scheduled. A single runqueue guarantees that the system is work-conserving and that it always schedules the  $m$  highest-priority tasks (if that many are available). In preparation of our analysis of APA scheduling, we summarize global scheduling as follows.

**Global Scheduling Invariant:** Let  $S(t)$  be the set of all tasks that are scheduled on any of the  $m$  processors at time  $t$ . Let  $prio(T_i)$  denote the priority of a task  $T_i$  at time  $t$ . If  $T_b$  is a backlogged task at time  $t$ , then under global scheduling:

$$\forall T_s \in S(t), prio(T_b) \leq prio(T_s) \wedge |S(t)| = m. \quad (3)$$

However, the Linux scheduler implements runqueues in a partitioned fashion. Therefore, to satisfy the global scheduling invariant, Linux requires explicitly triggered *migrations* so that a task is scheduled as soon as at least one of the processors is not executing a higher-priority task. These migrations are achieved by so-called “push” and “pull” operations, which are source-initiated and target-initiated migrations, respectively, as described next.

Let  $\Pi_s$  denote the source and let  $\Pi_t$  denote the target processor, and let  $T_m$  be the task to be migrated. A *push* operation is performed by  $\Pi_s$  on  $T_m$  if  $T_m$  becomes available for execution on  $\Pi_s$ 's runqueue (e.g., when a new job of  $T_m$  arrives, when a job of  $T_m$  resumes from suspension, or when a job of  $T_m$  is preempted by a higher priority job). The push operation iterates over runqueues of all processors and tries to identify the best runqueue (belonging to the target processor  $\Pi_t$ ) such that the task currently assigned to  $\Pi_t$  has a lower priority than  $T_m$ .

In contrast to a push operation, a *pull* operation is a target-initiated migration carried out by processor  $\Pi_t$  when it is about to schedule a job of priority lower than the previously scheduled task (e.g., when the previous job of a higher-priority task suspended or completed). The pull operation scans each processor  $\Pi_s$  for a task  $T_m$  assigned to  $\Pi_s$ 's runqueue such that  $T_m$  is backlogged and  $T_m$ 's priority exceeds that of all local tasks in  $\Pi_t$ 's runqueue. When multiple candidate tasks such as  $T_m$  are available for migration, the pull operation selects the task with the highest priority.

Preemptions are enacted as follows in Linux. Suppose a processor  $\Pi_s$  is currently serving a low-priority task  $T_l$  when a higher-priority task  $T_h$  becomes available for execution on  $\Pi_s$  (i.e., processor  $\Pi_s$  handles the interrupt that causes  $T_h$  to release a job). Then  $\Pi_s$  immediately schedules  $T_h$  instead of  $T_l$  and invokes a push operation on  $T_l$  to determine if  $T_l$  can be scheduled elsewhere. If no suitable migration target  $\Pi_t$  exists for  $T_l$  at the time of preemption,  $T_l$  will remain queued on  $\Pi_s$  until it is discovered later by a pull operation (or until  $T_h$ 's job completes and  $\Pi_s$  becomes available again).

Crucially, a push operation is triggered only for tasks that are *not* currently scheduled, and a pull operation similarly never migrates a task that is already scheduled. Thus, once a task is scheduled on a processor  $\Pi_t$ , it can only be “dislodged” by the arrival of a higher-priority task on  $\Pi_t$ , either due to

a push operation targeting  $\Pi_t$  or due to an interrupt handled by  $\Pi_t$ . On which processor a job is released depends on the specific interrupt source (e.g., timers, I/O devices, etc.), and how the interrupt routing is configured in the multiprocessor platform (e.g., interrupts could be routed to a specific processor or distributed among all processors). We make no assumptions on which processor handles interrupts, that is, we assume that a job may be released on potentially any processor.

### B. APA Scheduling

APA scheduling is similar to global scheduling in that a task may have to be migrated to be scheduled. Under global scheduling, a task is allowed to migrate to any processor in the system, whereas under APA scheduling, a task is allowed to migrate only to processors included in its processor affinity set. Therefore, APA scheduling provides a slightly different guarantee than the global scheduling invariant.

**APA Scheduling Invariant:** Let  $T_b$  be a backlogged task at time  $t$  with processor affinity  $\alpha_b$ . Let  $S'(t)$  be the set of tasks that are scheduled on any processors in  $\alpha_b$  at time  $t$ . If  $\text{prio}(T_i)$  denotes the priority of a task  $T_i$  at time  $t$ , then under APA scheduling:

$$\forall T_s \in S'(t), \text{prio}(T_b) \leq \text{prio}(T_s) \wedge |S'(t)| = |\alpha_b|. \quad (4)$$

A key feature of Linux’s scheduler is that push and pull operations seamlessly generalize to APA scheduling. A push operation on  $\Pi_s$  migrates  $T_m$  from  $\Pi_s$  to  $\Pi_t$  only if  $\Pi_t \in \alpha_m$ . Similarly, a pull operation on  $\Pi_t$  pulls  $T_m$  from  $\Pi_s$  only if  $\Pi_t \in \alpha_m$ . In short, the two operations never violate a task’s processor affinity when it is migrated.

The push and pull operations together ensure that a task  $T_m$  is waiting to be scheduled only if all processors in  $\alpha_m$  are busy executing higher-priority tasks. However, as discussed above, note that push and pull operations never migrate already scheduled, higher-priority tasks to “make room” for  $T_m$ . As a result,  $T_m$  may remain backlogged if all processors in  $\alpha_m$  are occupied by higher-priority tasks, even if some task  $T_h \in S'(t)$  could be scheduled on another processor  $\Pi_x$  not part of  $\alpha_m$  (i.e., in the worst case, if  $\Pi_x \in \alpha_h$  and  $\Pi_x \notin \alpha_m$ , then  $\Pi_x$  may idle while  $T_m$  is backlogged). For instance, such a scenario may occur if  $T_h$  is released on the processor that  $T_m$  is scheduled on since Linux switches immediately to higher-priority tasks and only then attempts to push the preempted task. While this approach may not be ideal from a schedulability point of view, it has the advantage of simplifying the implementation; we revisit this issue in Sec. VI.

From the definitions of the global and APA scheduling invariants, we can infer that global scheduling is a special case of APA scheduling, where all tasks have an affinity  $\alpha_i = \pi$ . Conversely, APA scheduling is more general than global scheduling, but also “global-like” from the point of view of backlogged task—a task is only backlogged if “all available” processors are serving higher-priority tasks. We discuss this idea in detail in the next sections. We begin by showing APA JLFP scheduling to strictly dominate global, clustered, and partitioned

JLFP scheduling in Sec. III below, and then present in Sec. IV general schedulability tests applicable to all schedulers that guarantee the APA scheduling invariant given in Equation 4.

### III. GENERALITY OF APA SCHEDULING

Recall from Sec. I that a careful assignment of processor affinities can improve throughput, can simplify load balancing (e.g., to satisfy thermal constraints), and can be used to isolate applications from each other (e.g., for security reasons). In this section, we weigh the schedulability benefits of APA scheduling against global and partitioned scheduling and show that APAs are a useful construct from the point of view of real-time scheduling as well.

As discussed in Sec. I, APA scheduling is a constrained-migration model that limits the scheduling and migration of a task to an arbitrary set of processors. By assigning an appropriate processor affinity, a task can either be allowed to migrate among all processors (like global scheduling), allowed to migrate among a subset of processors (like clustered scheduling), or not allowed to migrate at all (like partitioned scheduling).

**Lemma 1:** A task set that is schedulable under global, partitioned, or clustered scheduling, is also schedulable under APA scheduling.

*Proof-sketch:* Trivial; APA scheduling can emulate global, clustered, and partitioned scheduling by assigning every task in the task set an appropriate processor affinity. ■

However, unlike under clustered scheduling, the processor affinities of tasks under APA scheduling need not be disjoint, i.e., two tasks  $T_i$  and  $T_k$  can have non-equal processor affinities  $\alpha_i$  and  $\alpha_k$  such that  $\alpha_i \cap \alpha_k \neq \emptyset$ . As a result, there exist task sets that are schedulable under APA scheduling, but infeasible under global, clustered, and partitioned scheduling.

**Theorem 1:** APA JLFP scheduling strictly dominates global, partitioned, and clustered JLFP scheduling.

*Proof:* We show that there exists a task set which is schedulable under APA scheduling, but not schedulable under global, partitioned, or clustered JLFP scheduling. Consider the task set described in Table I, which is to be scheduled on two processors. Consider any JLFP rule to prioritize tasks and an asynchronous arrival sequence, where task  $T_2$  arrives at time 1 but all other tasks arrive at time 0. We try to schedule this task set using global, partitioned, and APA JLFP scheduling. We do not explicitly consider clustered scheduling because, for two processors, clustered scheduling reduces to either global or partitioned scheduling.

*Global scheduling:* Since tasks  $T_1$  and  $T_2$  have unit densities each and there are two processors in the system, to obtain a schedule without any deadline misses, jobs of these tasks must always have the two highest priorities (although their relative priority ordering may differ under different JLFP policies). Also, since the deadlines of tasks  $T_3$  and  $T_4$  are very small compared to the execution costs of tasks  $T_5$ ,  $T_6$ , and  $T_7$ , jobs of tasks  $T_3$  and  $T_4$  must be assigned higher priorities relative to the jobs of tasks  $T_5$ ,  $T_6$ , and  $T_7$ . Due to these constraints, either jobs of  $T_3$  must be assigned the third-highest priority and

TABLE I

Task	$e_i$	$d_i$	$p_i$
$T_1$	1	1	10,000
$T_2$	2	2	10,000
$T_3$	3	4	10,000
$T_4$	2	4	10,000
$T_5$	501	1,000	1,000
$T_6$	5,001	10,000	10,000
$T_7$	5,000	10,000	10,000

TABLE II

Task	$\alpha_i$
$T_1$	$\{\Pi_1\}$
$T_2$	$\{\Pi_2\}$
$T_3$	$\{\Pi_1\}$
$T_4$	$\{\Pi_2\}$
$T_5$	$\{\Pi_1\}$
$T_6$	$\{\Pi_2\}$
$T_7$	$\{\Pi_1, \Pi_2\}$

jobs of  $T_4$  the fourth-highest priority, or vice versa. In either case, either  $T_3$  or  $T_4$  (whichever has the job with the lower priority) misses its deadline because neither can exploit the parallelism during  $[3, 4)$ , as illustrated in Figure 1. Therefore, *the task set is infeasible under global JLFP scheduling with any JLFP rule.*

*Partitioned scheduling:* A feasible partition must have a total utilization of at most one. The utilizations of tasks  $T_5$ ,  $T_6$ , and  $T_7$  are 0.501, 0.5001, and 0.5 respectively. Clearly, these three tasks cannot be partitioned into two bins, each with total utilization at most one. Therefore, *the task set cannot be partitioned onto a two-processor system.*

*APA scheduling:* The failure of global scheduling suggests that tasks  $T_3$  and  $T_4$  (and also tasks  $T_1$  and  $T_2$  because of their unit densities) should be restricted to separate processors. This separation cannot be achieved by partitioning as tasks  $T_5$ ,  $T_6$ , and  $T_7$  prevent successful partitioning of the task set. Therefore, using processor affinities as given in Table II, we partition tasks  $T_1, T_2, T_3, T_4, T_5$ , and  $T_6$  but allow task  $T_7$  to migrate. The task set is now schedulable assuming FP as the JLFP rule (lower indices imply higher priorities). To show this, we next prove the schedulability of task  $T_7$  (tasks  $T_1, T_2, T_3, T_4, T_5$ , and  $T_6$  can be trivially shown to be schedulable using uniprocessor response-time analysis). Consider an interval  $\Gamma = [t_a, t_d)$  of length 10,000 where  $t_a$  is the arrival time and  $t_d$  is the absolute deadline of a job  $J_7$  belonging to task  $T_7$ . We look at the two processors  $\Pi_2$  and  $\Pi_1$  in sequence. First, we bound the minimum time for which  $J_7$  can execute on  $\Pi_2$ . Then, to ensure  $T_7$ 's schedulability, we argue that  $J_7$  can always satisfy its remaining processor demand on  $\Pi_1$ .

We use techniques from [7] to bound the maximum interference such that any demand due to a carry-in job (i.e., a job released prior to  $t_a$ ) is also accounted for. During  $\Gamma$ , the maximum interference incurred by  $J_7$  on  $\Pi_2$  due to tasks  $T_2, T_4$ , and  $T_6$  is bounded by  $2 + 2 + 5001 = 5005$  (the exact interference  $\mathcal{I}$  varies with the inter-arrival times of jobs). If  $\mathcal{I} \leq 5000$ , then  $J_7$  can be scheduled successfully on  $\Pi_2$  itself. However, if  $\mathcal{I} > 5000$ ,  $J_7$  must satisfy its remaining demand on  $\Pi_1$ , i.e., if  $\mathcal{I} = 5000 + \delta$  where  $\delta \in [1, 5]$ , then  $J_7$  must execute on processor  $\Pi_1$  for at least  $\delta$  time units.

Let  $\Gamma'$  denote the cumulative interval(s) in  $\Gamma$  when jobs of  $T_6$  interfere with  $J_7$  on  $\Pi_2$ . Since the contribution of tasks  $T_2$  and  $T_4$  to  $\mathcal{I}$  is at most  $2 + 2 = 4$ , the contribution of  $T_6$  to  $\mathcal{I}$  is at least  $4996 + \delta$  (recall that  $\mathcal{I} = 5000 + \delta$ ). This contribution is a result of either one job or two consecutive jobs of  $T_6$  (in case the release of the first job of  $T_6$  does not

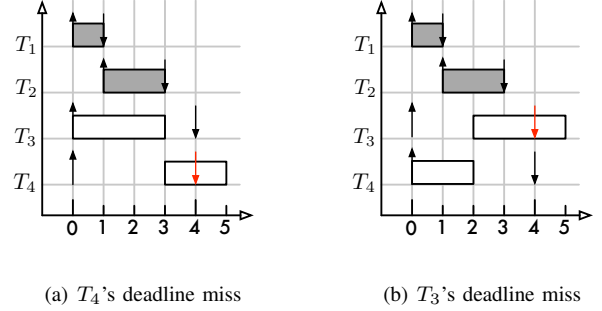


Fig. 1. Global JLFP schedules of tasks  $T_1, T_2, T_3$ , and  $T_4$  in Table I. The small up-arrows and down-arrows indicate job-arrivals and deadlines respectively. Jobs executing on processor  $\Pi_1$  are in shaded in grey and jobs executing on processor  $\Pi_2$  are shaded in white. (a)  $T_3$ 's job is assigned a higher priority than  $T_4$ 's job and consequently  $T_4$ 's job misses its deadline. (b)  $T_4$ 's job is assigned a higher priority than  $T_3$ 's job and consequently  $T_3$ 's job misses its deadline.

align with  $t_a$  but precedes  $t_a$ ). In either case,  $\Gamma'$  consists of at least one contiguous interval  $\Gamma'' \in \Gamma'$  of length  $2498 + \delta/2$ . However, in any contiguous interval of length  $2498 + \delta/2$ ,  $\Pi_1$  can be busy executing jobs of tasks  $T_1, T_3$ , and  $T_5$  for at most  $\lceil (2498 + \delta/2)/1000 \rceil * 501 + 2 + 2 = 1507$  time units, i.e., while  $\Pi_2$  is continuously unavailable during  $\Gamma''$ ,  $\Pi_1$  is available for at least  $2498 + \delta - 1507 = 991 + \delta \gg \delta$  time units, and consequently  $J_7$  has enough opportunities to finish its remaining execution on  $\Pi_1$ . Therefore,  $T_7$  is schedulable and *the task set is schedulable under APA JLFP scheduling.* ■

Theorem 1 establishes the dominance of APA scheduling with a JLFP policy over global, partitioned, and clustered JLFP scheduling and provides further motivation to explore this scheduling technique. Concerning the more general case of JLDP policies, we leave the dominance question for future work. However, we note that there may be a case of equivalence since there exist global JLDP schedulers that are optimal for implicit-deadline tasks [9], and since optimal online scheduling of arbitrary-deadline tasks is generally impossible [20]. In any case, Lemma 1 shows that APA scheduling is at least as powerful as global, clustered, and partitioned scheduling combined even w.r.t. JLDP policies.

In the next section, we provide initial schedulability analysis techniques for APA schedulers with any JLFP policy.

#### IV. SCHEDULABILITY ANALYSIS<sup>3</sup>

There are many variants of APA schedulers deployed in current real-time operating systems such as VxWorks, LynxOS, QNX, and real-time variants of Linux. However, to the best of our knowledge, no schedulability analysis test applicable to task sets with APAs has been proposed to date. In this section, we apply the ideas from Sec. II that relate APA scheduling to the well-studied global scheduling problem, and propose simple and efficient techniques to analyze task sets for APA

<sup>3</sup>Correction: Lemmas 2 and 4 are stated in too general terms and do not hold in all cases. Refer to Appendix A for details.

scheduling. In a nutshell, we reduce APA scheduling to “global-like” subproblems, which allows reuse of the large body of literature on global schedulability analysis.<sup>4</sup> The section is divided into three parts. We start with a simple method for analyzing task sets with APAs using tests for global scheduling and argue its correctness. The second part introduces a more robust test with reduced pessimism, but at the cost of high computational complexity. The last part introduces a heuristic-based test to balance the cost versus pessimism tradeoff by considering only “promising” subproblems.

### A. Reduction to Subproblems

Recall from Sections II and III that, for a given task  $T_i$ , global scheduling is a special case of APA scheduling when  $\alpha_i = \pi$ . Similarly, for a subproblem with a reduced processor set  $\alpha_i$ , and a reduced task set  $tasks(\alpha_i)$ , APA scheduling reduces to global scheduling.<sup>5</sup> For example, consider the scheduling problems illustrated in Figure 2. Figure 2(a) represents an APA scheduling problem, where each task has an individual processor affinity. Figure 2(b) represents a subproblem of the former problem that is also an APA scheduling problem. However, as in a global scheduling problem, task  $T_5$ 's processor affinity spans all the processors in this subproblem. Also, all the tasks in this subproblem can interfere with  $T_5$ . Therefore, the subproblem is global w.r.t.  $T_5$ . In other words, if  $T_5$  is schedulable using global scheduling on a platform consisting only of the processors in  $\alpha_5$ , then it is also schedulable using APA scheduling on the processor platform  $\pi$ .<sup>6</sup> This idea is formally stated in the lemma below for JLFP schedulers and thus also extends to FP scheduling. Recall that  $tasks(\rho)$  denotes the set of tasks that can be scheduled on at least one processor in  $\rho$ .

**Lemma 2:** *If a task  $T_i \in tasks(\alpha_i)$  is schedulable when the reduced task set  $tasks(\alpha_i)$  is globally scheduled on the reduced processor platform  $\alpha_i$  using a JLFP policy  $A$ , then  $T_i$  is also schedulable under APA scheduling of  $\tau$  on the processor platform  $\pi$  using the same JLFP policy  $A$ .<sup>7</sup>*

*Proof sketch:* Suppose not. Then a task  $T_i \in tasks(\alpha_i)$  is schedulable under global scheduling on the processor platform  $\alpha_i$  using a JLFP policy  $A$ , but it is not schedulable under APA scheduling on the processor platform  $\pi$  using the same JLFP policy  $A$ . For a job  $J_i$  of any task  $T_i$  to miss its deadline, its response time  $r_i$  must be greater than its deadline, i.e.,  $r_i > d_i$ , where  $r_i$  is the sum of  $T_i$ 's WCET and the time during which  $J_i$  was interfered with by other tasks.

Task  $T_i$  incurs interference whenever all processors on which  $T_i$  can be scheduled (i.e.,  $\alpha_i$ ) are busy executing

<sup>4</sup>**Correction:** the proposed approach allows reuse of certain techniques, but not of the entire body of literature on global schedulability analysis.

<sup>5</sup>**Correction:** APA scheduling resembles global scheduling in some regards, but it does not formally reduce to global scheduling.

<sup>6</sup>**Correction:** as demonstrated in Appendix A.1, this is not generally true, as the interference due to higher-priority jobs under APA and global scheduling may differ even if the analyzed job's affinity set includes all processors.

<sup>7</sup>**Correction:** while certain sufficient (but not necessary) global schedulability tests can indeed be applied to global-like subproblems to assess the schedulability of a task under APA scheduling (e.g., see Appendix B), the claim does not hold in general; see Appendix A.1 for a counterexample.

tasks other than  $T_i$ . With respect to a given interval  $[t_1, t_2)$ , let  $\Theta_i(t_1, t_2)$  denote the sub-interval (or a union of non-contiguous sub-intervals) during which all processors in  $\alpha_i$  are busy executing tasks other than  $T_i$ . Therefore, if  $|\Theta_i(t_1, t_2)|$  represents the cumulative length of the sub-intervals denoted by  $\Theta_i(t_1, t_2)$ , then for a job  $J_i$  arriving at  $t_a$  to miss its deadline, it is necessary that  $e_i + |\Theta_i(t_a, t_a + d_i)| > d_i$ .

Since  $T_i$  is not schedulable under APA scheduling on the processor platform  $\pi$ , there exists an arrival sequence and a corresponding interval  $[t_a, t_d)$  of length  $d_i$  such that a job  $J_i^{APA}$  of  $T_i$  arrives at time  $t_a$  and misses its deadline at time  $t_d$  under APA scheduling, i.e.,

~~$$\exists t_a : e_i + |\Theta_i^{APA}(t_a, t_d)| > d_i. \quad (5)$$~~

However, since  $T_i \in \tau$  is schedulable under global scheduling on the reduced processor platform  $\alpha_i$ , for any possible arrival sequence and a corresponding interval  $[t_a, t_d)$  of length  $d_i$ , a job  $J_i^G$  of  $T_i$  arriving at  $t_a$  successfully completes its execution before  $t_d$ , i.e.,

~~$$\forall t_a : e_i + |\Theta_i^G(t_a, t_d)| \leq d_i. \quad (6)$$~~

The work that comprises  $\Theta_i^{APA}(t_a, t_d)$  is computed upon  $\alpha_i$ .  $\Theta_i^G(t_a, t_d)$  is computed upon all processors in the processor platform, which is equal to  $\alpha_i$  in this case. Also, by construction, under both APA and global scheduling, the same set of tasks keeps processors in  $\alpha_i$  busy during  $\Theta_i^{APA}(t_a, t_d)$  and  $\Theta_i^G(t_a, t_d)$ , i.e., the set of possible arrival sequences are equivalent. Therefore, if there exists an interval  $[t_a, t_d)$  such that  $\Theta_i^{APA}(t_a, t_d)$  exceeds  $d_i - e_i$ , then there exists such an interval for  $\Theta_i^G(t_a, t_d)$  as well, and Equations 5 and 6 cannot both be true simultaneously. ■

Using the equivalence from Lemma 2, we design a simple schedulability test for APA scheduling based on global schedulability analysis. In this paper, our focus is on global tests in general, that is, we do not focus on any particular test.<sup>8</sup> For this purpose, we assume the availability of a generic<sup>9</sup> test  $GlobalAnalysis(A, T_i, \pi, \zeta_i)$  to analyze the schedulability of a single task, where  $A$  is the scheduling policy,  $T_i$  is the task to be analyzed,  $\pi$  is the processor set on which the task is to be scheduled, and  $\zeta_i$  is the set of tasks that can interfere with  $T_i$ . The test returns *true* if  $T_i$  is schedulable and *false* otherwise. Note that a result of *true* does not imply that all tasks in  $\tau$  are schedulable; we are only concerned with schedulability of task  $T_i$ . Using this interface we define a simple method to analyze tasks sets for APA scheduling. The key idea is to identify for each task an “equivalent” global subproblem, and to then invoke  $GlobalAnalysis(A, T_i, \pi, \zeta_i)$  on that subproblem.

**Lemma 3:** *A task set  $\tau$  is schedulable on a processor set*

<sup>8</sup>**Correction:** since Lemma 2 is incorrect, the approach described in this section is not applicable to all global schedulability tests.

<sup>9</sup>**Correction:** assume a *specific* test, namely one of the tests proven to be compatible in Appendix B.



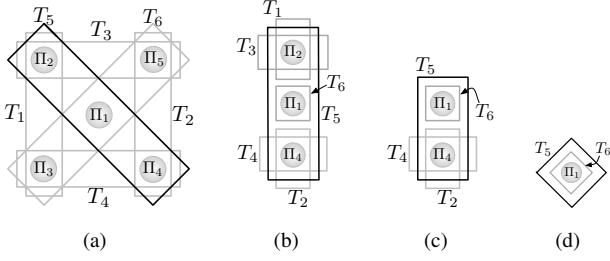


Fig. 2. Four scheduling problems (a), (b), (c), and (d) are illustrated here. The circles represent the processors and the rectangles represent the tasks and their associated processor affinities, e.g., problem (a) consists of the processor set  $\pi = \{\Pi_1, \Pi_2, \Pi_3, \Pi_4, \Pi_5\}$  and the task set  $\tau = \{T_1, T_2, T_3, T_4, T_5, T_6\}$ . Problem (b), (c), and (d) are subproblems of problem (a). Note that all the subproblems are global w.r.t. task  $T_5$ , i.e., like in a global scheduling problem,  $T_5$  can be scheduled on all processors in these subproblems and all tasks in these subproblems can potentially interfere with  $T_5$ .

$\pi$  under APA scheduling using a JLFP policy A if

$$\bigwedge_{\forall T_i \in \tau} \text{GlobalAnalysis}(A, T_i, \alpha_i, I_i^A). \quad (7)$$

*Proof sketch:* The analysis checks schedulability of each task  $T_i \in \tau$  under global scheduling on processor platform  $\alpha_i$ . From Lemma 2, if each task  $T_i \in \tau$  is schedulable on the corresponding reduced processor platform  $\alpha_i$ , then  $T_i$  is also schedulable on the processor platform  $\pi$  under APA scheduling.<sup>10</sup> Therefore, the entire task set  $\tau$  is schedulable under APA scheduling with policy A on the processor platform  $\pi$ . ■

The analysis technique in Lemma 3 is a straightforward way to reuse global schedulability analysis for analyzing task sets with APAs, i.e., task sets to be scheduled by APA scheduling. Apart from the computations required by a conventional global schedulability test, this new analysis technique requires only minor additions for computing the interfering task set (e.g.  $I_i^{FP}$  for an FP rule,  $I_i^{EDF}$  for an EDF rule) for every task  $T_i$  on the respective processor platform  $\alpha_i$ . However, this algorithm assumes that the processors in overlapping affinity regions must service the demand of all tasks that can be scheduled in that overlapping region. Therefore, it is possible that a schedulability test claims task  $T_i$  to be not schedulable with the given processor affinity  $\alpha_i$ , but claims it to be schedulable with a different processor affinity  $\alpha'_i \subset \alpha_i$ , i.e., the result of the schedulability analysis in Lemma 3 may vary for the same task if reduced to different subproblems.

**Example 1:** Consider the task set described in Table III, which is to be scheduled on five processors under APA scheduling. The processor affinities of the tasks are also illustrated in Figure 2(a). Assume a fixed-priority scheduler with the following priority scheme:  $\forall i < k, \text{prio}(T_i) > \text{prio}(T_k)$ . To analyze the schedulability of the task set, we define  $\text{GlobalAnalysis}(FP, T_i, \alpha_i, I_i^{FP})$  in Lemma 3 as a modified version of the response-time analysis for global fixed-priority scheduling in [10] (see Sec. V for details). Task  $T_5$  fails the  $\text{GlobalAnalysis}(FP, T_5, \alpha_5, I_5^{FP})$  test with the given processor affinity (see Figure 2(b) for the corresponding

<sup>10</sup>**Correction:** this property must be established for each global schedulability test that is to be used with the proposed reduction approach; see Appendix B.

TABLE III

Task	$e_i$	$d_i$	$p_i$	$\alpha_i$
$T_1$	5	6	6	$\{\Pi_2, \Pi_3\}$
$T_2$	3	4	4	$\{\Pi_4, \Pi_5\}$
$T_3$	1	4	4	$\{\Pi_2, \Pi_5\}$
$T_4$	2	8	8	$\{\Pi_3, \Pi_4\}$
$T_5$	2	5	12	$\{\Pi_1, \Pi_2, \Pi_4\}$
$T_6$	1	3	12	$\{\Pi_1, \Pi_3, \Pi_5\}$

subproblem), i.e.  $\alpha_5 = \{\Pi_1, \Pi_2, \Pi_4\}$ . However, if applied to a different subset of the processor affinity (Figure 2(c)), i.e.,  $\alpha'_5 = \{\Pi_1, \Pi_4\}$ ,  $T_5$  is deemed schedulable by the test. Note that on the processor platform  $\alpha_5$ , all four higher priority tasks can interfere with  $T_5$ ; but on the processor platform  $\alpha'_5$ , only  $T_2$  and  $T_4$  can interfere with  $T_5$ . Therefore, there is a significant reduction in the total interference on  $T_5$ , and consequently the test claims  $T_5$  to be schedulable on  $\alpha'_5$ , but not on  $\alpha_5$ .

In the next section, we use Example 1 to motivate an analysis technique for APA scheduling that checks schedulability of a task  $T_i$  on all possible subsets of  $\alpha_i$ . We also argue the formal correctness of this approach by proving that schedulability of  $T_i$  on a processor platform  $\alpha'_i \subset \alpha_i$  implies schedulability of  $T_i$  on the processor platform  $\alpha_i$ .

### B. Exhaustive Reduction

**Lemma 4:** If a task  $T_i \in \tau$  is schedulable under APA scheduling with the processor affinity  $\alpha'_i \subset \alpha_i$  and task set  $\tau$ , then  $T_i$  is also schedulable under APA scheduling with the affinity  $\alpha_i$  and task set  $\tau$ .<sup>11</sup>

*Proof sketch:* We prove the lemma by contradiction, analogous to Lemma 2. Recall from the proof of Lemma 2 that  $\Theta_i(t_1, t_2)$  denotes the sub-interval of interference during which all processors in  $\alpha_i$  are busy executing tasks other than  $T_i$ . Similarly, we define  $\Theta'_i(t_1, t_2)$  over all processors in  $\alpha'_i$ . We assume that  $T_i$  is not schedulable under APA scheduling with processor affinity  $\alpha_i$  and task set  $\tau$  but  $T_i$  is schedulable under APA scheduling with the processor affinity  $\alpha'_i \subset \alpha_i$ , i.e., if  $t_a$  is the arrival-time of a job of  $T_i$  for an arbitrary job arrival sequence, then

~~$$\exists t_a : e_i + |\Theta_i(t_a, t_a + d_i)| > d_i, \quad (8)$$~~

~~$$\forall t_a : e_i + |\Theta'_i(t_a, t_a + d_i)| \leq d_i. \quad (9)$$~~

For any arbitrary, fixed arrival sequence, at any time instance, if all processors in  $\alpha_i$  are busy executing tasks other than  $T_i$ , then all processors in  $\alpha'_i$  must also be executing tasks other than  $T_i$  since  $\alpha'_i \subset \alpha_i$ . Thus,  $\Theta'_i(t_a, t_a + d_i)$  is a superset ( $\supseteq$ ) of  $\Theta_i(t_a, t_a + d_i)$ , and hence,  $|\Theta'_i(t_a, t_a + d_i)| \geq |\Theta_i(t_a, t_a + d_i)|$ . Equations 8 and 9 cannot be true simultaneously. ■

In Example 1, the schedulability test could not claim task  $T_5$  to be schedulable with a processor affinity of  $\alpha_5$ . However, the test claimed that the same task  $T_5$ , assuming a reduced

<sup>11</sup>**Correction:** while the claim holds under FP scheduling, it does not hold for all JLFP policies; see Appendix A.2 for a counterexample using EDF.

processor affinity of  $\alpha'_5 \subset \alpha_5$ , is schedulable. Note that this example does not contradict Lemma 4. While the result of Lemma 4 pertains to actual schedulability under APA scheduling, the schedulability test used in Example 1 is a sufficient, but not necessary, test, which is subject to inherent pessimism, both due to the subproblem reduction and because the underlying global schedulability test is only sufficient, but not necessary, as well. Therefore, it may return negative results for tasks that are actually schedulable under APA scheduling.

We next present a schedulability analysis for APA scheduling based on Lemma 4 and the simple test in Lemma 3 that exploits the observation that it can be beneficial to consider only a subset of a task’s processor affinity. In this method, global schedulability analysis is performed for a task  $T_i \in \tau$  on all possible subsets of its processor affinity, i.e.,  $\forall S \subseteq \alpha_i$ . The task  $T_i$  is deemed schedulable if it passes the analysis for at least one such subset  $S$ , and the task set  $\tau$  is deemed schedulable if all tasks  $T_i \in \tau$  pass the test. Recall from Lemma 4 that schedulability of a task  $T_i$  under APA scheduling with processor affinity  $S \subseteq \alpha_i$  implies schedulability of  $T_i$  under APA scheduling with processor affinity  $\alpha_i$ , however, it does not require modifying the processor affinity of  $T_i$  from  $\alpha_i$  to  $S$ . In particular, while analyzing schedulability of any task  $T_i$ , the processor affinities of other tasks remain unchanged.

**Lemma 5:** *A task set  $\tau$  is schedulable on a processor set  $\pi$  under APA scheduling using a JLFP policy  $A$  if*

$$\bigwedge_{\forall T_i \in \tau} \left( \bigvee_{\forall S_i \subseteq \alpha_i} \text{GlobalAnalysis}(A, T_i, S_i, I_i^A \cap \text{tasks}(S_i)) \right). \quad (10)$$

*Proof sketch:* If there exists a subset  $S_i \subseteq \alpha_i$  such that  $T_i$  is schedulable using global scheduling on processor platform  $S_i$  using a JLFP policy  $A$ , then from Lemma 2,  $T_i$  is also schedulable under APA scheduling with the processor affinity  $S_i$  and the policy  $A$ . From Lemma 4, since  $T_i$  is schedulable under APA scheduling with the processor affinity  $S_i \subseteq \alpha_i$ ,  $T_i$  is also schedulable under APA scheduling with the processor affinity  $\alpha_i$ .<sup>12</sup> Therefore, if corresponding subsets exist for every task in  $\tau$ , the task set  $\tau$  is schedulable on the processor set  $\pi$  under APA scheduling using JLFP policy  $A$ . ■

The schedulability test given by the above lemma requires iterating over potentially every subset  $S \subseteq \alpha_i$ . This makes the algorithm robust in the sense that it eliminates all false negatives that occur when a task  $T_i$  can be claimed to be schedulable only on a subset of its processor affinity  $S \subset \alpha_i$ , but not on its processor affinity  $\alpha_i$ . However, since  $|\alpha_i|$  is bounded by  $m$ , and since the schedulability tests have to be run for all the tasks in the task set, in the worst case, the algorithm requires  $O(n \cdot 2^m)$  invocations of  $\text{GlobalAnalysis}(A, T_i, \alpha_i, I_i^A)$ . Despite the exponential complexity, an exhaustive approach is still feasible for contemporary embedded multiprocessors with up to eight processors. However, for multiprocessor systems with higher number of processors, we need an alternative algorithm that does not analyze a task for all possible subsets of its processor

<sup>12</sup>**Correction:** this property must be established for each global schedulability test that is to be used with the proposed reduction approach; see Appendix B.

affinity. Instead, in the next section, we propose a heuristic to identify and test only a few “promising” subsets.

### C. Heuristic-based Reduction

We propose a heuristic that helps to choose promising subsets of a task’s processor affinity to test the task’s schedulability. The heuristic removes one or a few processors at a time from the task’s processor affinity such that maximum benefit is achieved in terms of the interference lost (i.e., the processor time gained). We illustrate this intuition with an example below and then proceed with a detailed explanation of the heuristic and the new analysis technique.

**Example 2:** Consider the task set from Example 1 (Table III). Since the schedulability of tasks  $T_1, T_2, \dots, T_5$  has already been established in Example 1, we carry out analysis for task  $T_6$  in this example.  $T_6$  fails  $\text{GlobalAnalysis}(FP, T_6, \alpha_6, I_6^{FP})$  with the processor affinity as given in Figure 2(b), i.e.,  $\alpha_6 = \{\Pi_1, \Pi_3, \Pi_5\}$ . Therefore, we find an appropriate subset  $\alpha'_6 \subset \alpha_6$  such that  $T_6$  is claimed to be schedulable on processor platform  $\alpha'_6$ . However, unlike the algorithm given in Lemma 5, we intelligently select only the promising subsets of  $\alpha_6$ . In each iteration, we remove the processor that contributes the most to the total interference w.r.t.  $T_6$ .

**Iteration 1:**  $\alpha_6 = \{T_1, T_3, T_5\}$ . The removal candidates in  $\alpha_6$  are  $\Pi_1, \Pi_3$  and  $\Pi_5$ . Removing  $\Pi_1$  leads to removal of  $\{T_5\}$ , removing  $\Pi_3$  leads to removal of  $\{T_1, T_4\}$  and removing  $\Pi_5$  leads to removal of  $\{T_2, T_3\}$  from  $I_6^{FP}$ . We remove  $\{\Pi_3\}$  because  $\{T_1, T_4\}$  contributes most to the total interference on  $T_6$ . But  $T_6$  still fails the schedulability test.

**Iteration 2:**  $\alpha'_6 = \{\Pi_1, \Pi_5\}$ . The removal candidates in  $\alpha'_6$  are  $\Pi_1$  and  $\Pi_5$ . Removing  $\Pi_1$  leads to removal of  $\{T_5\}$  and removing  $\Pi_5$  leads to removal of  $\{T_2, T_3\}$  from  $I_6^{FP}$ . We choose to remove  $\{\Pi_5\}$  because  $\{T_2, T_3\}$  contributes more to the total interference on  $T_6$  than  $\{T_5\}$ . The new subset is thus  $\alpha''_6 = \{\Pi_5\}$  and  $T_6$  passes the schedulability test. Therefore,  $T_6$  is schedulable under APA scheduling with an FP policy.

The intuition of iteratively removing processors from the processor affinity until the processor set is empty is formalized with the heuristic-based algorithm  $\text{HeuristicBasedAnalysis}(T_i, \alpha_i, I_i)$ , which is defined in Algorithm 1. With this procedure, we obtain a new schedulability analysis for APA scheduling: a task set  $\tau$  is schedulable under APA scheduling using JLFP if  $\forall T_i \in \tau$ ,  $\text{HeuristicBasedAnalysis}(T_i, \alpha_i, I_i)$  returns *true*.

Algorithm 1 shows the pseudo-code for heuristically determining subsets of  $\alpha_i$  and then invoking global analysis on those subsets.  $\alpha_i^k$  is the new subset to be analyzed in the beginning of the  $k^{\text{th}}$  iteration and  $I_i^k$  is the corresponding interfering task set.  $RC$  denotes the set of removal candidates. A removal candidate is a set of processors  $c$  such that, if  $c$  is removed from  $\alpha_i^k$  to obtain the new subset  $\alpha_i^{k+1}$ , then there is a non-zero decrease in the total interference on  $T_i$  from tasks in  $I_i^{k+1}$  (compared to the total interference on  $T_i$  from the tasks in  $I_i^k$ ). In other words, removing  $c$  from  $\alpha_i^k$  should lead



---

**Algorithm 1** *HeuristicBasedAnalysis*( $A, T_i, \alpha_i, I_i^A$ )

---

```
1:  $\alpha_i^0 \leftarrow \alpha_i$ 
2:  $I_i^0 \leftarrow I_i^A$ 
3:  $k \leftarrow 0$ 
4: repeat
5:   if GlobalAnalysis( $A, T_i, \alpha_i^k, I_i^k$ ) is true then
6:     return true
7:   end if
8:    $RC \leftarrow \phi$ 
9:   for all  $T_x \in I_i^k$  do
10:     $RC \leftarrow RC \cup \{\alpha_i^k \cap \alpha_x\}$ 
11:   end for
12:   for all  $c \in RC$  do
13:     $t(c) \leftarrow \text{tasks}(\alpha_i^k) \setminus \text{tasks}(\alpha_i^k \setminus c)$ 
14:     $\Delta(c) \leftarrow \sum_{T_x \in t(c)} \left( \left\lceil \frac{d_i}{p_x} \right\rceil + 1 \right) e_x$ 
15:   end for
16:    $c' \leftarrow c \in RC$  with largest  $\frac{\Delta(c)}{|c|}$  (tie break using  $|c|$ )
17:    $\alpha_i^{k+1} \leftarrow \alpha_i^k \setminus c'$ 
18:    $I_i^{k+1} \leftarrow I_i^k \setminus t(c')$ 
19: until  $(\alpha_i^{k+1} = \alpha_i^k) \vee (\alpha_i^{k+1} = \phi)$ 
```

---

to removal of at least one task from  $I_i^k$ . Let  $t(c)$  be the set of tasks removed from  $I_i^k$  if  $c$  is removed from  $\alpha_i^k$ . To select the “best” removal candidate, we use a metric that we call *estimated demand reduction per processor*, as defined below ( $\Delta(c)$  is computed in line 14 of Algorithm 1).

$$\frac{\Delta(c)}{|c|} = \frac{1}{|c|} \sum_{T_x \in t(c)} \left( \left\lceil \frac{d_i}{p_x} \right\rceil + 1 \right) e_x \quad (11)$$

For a removal candidate  $c$ , the estimated demand reduction per processor quantifies the approximate reduction in total interference after the  $k^{\text{th}}$  iteration, if  $c$  was removed from  $\alpha_i^k$  to obtain the new subset. The algorithm selects the removal candidate with the maximum estimated demand reduction per processor. In case of a tie between two or more candidates, we select the candidate with a smaller cardinality, e.g. among two candidates  $c', c'' \in RC$  with equal demand reduction per processor, we select  $c'$  if  $|c'| < |c''|$ . This ensures that more processors are available for scheduling  $T_i$  with the same amount of approximate total interference. We run this procedure iteratively either until we find a successful subset or until there is no change in  $\alpha_i^{k+1}$  w.r.t.  $\alpha_i^k$ .

The procedure *HeuristicBasedAnalysis*( $T_i, \alpha_i, I_i$ ) requires at most a number of iterations linear in the number of processors  $m$  because in every iteration at least one processor is removed from  $T_i$ 's processor affinity. Therefore, after at most  $|\alpha_i|$  iterations, the processor set becomes empty and the procedure terminates. The schedulability of a task set  $\tau$  requires each task  $T_i \in \tau$  to be schedulable. Therefore, in the worst case, this algorithm requires  $O(n \cdot m)$  invocations of *GlobalAnalysis*( $A, T_i, \alpha_i, I_i$ ). Compared to the exhaustive technique discussed in the previous section, this algorithm is much quicker to converge to a suitable subset. However, it is a heuristic-based algorithm and may still miss out on prospective subsets that may yield positive results. In the next section, we present results of schedulability experiments to validate the efficiency of the proposed analysis.

## V. EXPERIMENTS AND EVALUATION

We ran two sets of schedulability experiments, to compare global, partitioned, and APA FP scheduling, and to compare the performance of the proposed heuristic-based schedulability analysis for APA scheduling with the exhaustive analysis. We start with the discussion of the experimental setup and then report on the observed trends.

To perform the experiments, we generated task sets using Emberson *et al.*'s task set generator [19]. Motivated by our study of Linux, we restricted our focus to FP scheduling algorithms. For global FP scheduling, we used the DkC priority assignment heuristic [16] and the response-time analysis for global fixed-priority scheduling given in [10], which we denote as G-FP-RTA. For partitioned FP scheduling (P-FP), we used uniprocessor response-time time analysis [3] and assigned DM priorities. Partitioning was carried out using five standard bin-packing heuristics: *worst-fit-decreasing*, *first-fit-decreasing*, *best-fit decreasing*, *next-fit-decreasing*, and *almost-worst-fit-decreasing*; a task set was claimed schedulable under P-FP if it could be successfully partitioned using any of the heuristics and if each task in each partition passed the response-time test.

To implement *GlobalAnalysis*( $FP, T_i, \alpha_i, I_i^{FP}$ ) for APA scheduling, we used a modified version of G-FP-RTA, which we refer to as G-FP-APA. Note that the task sets used in APA scheduling experiments were assigned processor affinities using a heuristic similar to the one discussed in Sec. IV, i.e., we started with a global assignment and allowed shrinking of the processor affinities till a schedulable partitioned assignment was found, or till a schedulable arbitrary assignment (intermediate of global and partitioned assignments) was achieved. Since optimal priority assignment for APA scheduling is still an open problem, we tried using both DkC and DM priorities with the aforementioned heuristic. Also, tasks with a singleton processor affinity set were analyzed using uniprocessor response time analysis for efficiency (instead of G-FP-RTA).

We considered two variants of G-FP-APA, the exhaustive approach based on Lemma 5 (G-FP-APAE) and the heuristic-based approach based on Algorithm 1 (G-FP-APAh). For the first set of experiments, we varied the number of processors  $m$  from 3 to 8. Herein, we focus on graphs corresponding to  $m = 4$ ,  $m = 6$ , and  $m = 8$ . For the second set of experiments,  $m$  ranges from 3 to 5. We also varied the utilization from 0 to  $m$  in steps of 0.25 (excluding both end points). For each value of  $m$  and utilization  $u$ , we generated and tested 640 task sets, with a number of tasks ranging from  $m + 1$  to  $2.5m$ , in steps of 4. The periods of tasks were randomly chosen from [10ms, 100ms] following a log-uniform distribution. We summarize the main trends apparent in our results below.

*Experiment 1 (G-FP-APAE vs. G-FP-RTA vs. P-FP):* Each graph in Figure 3 consists of three curves, one for each of the three configurations, which represent the fraction of task sets schedulable as a function of the total system utilization. For utilization greater than 75%, G-FP-APAE performs consistently better than P-FP, though the average improvement is modest, in the range of 0%-10%. From a schedulability point of view,

we expect APA scheduling to provide the most benefit for task sets that cannot be partitioned easily, nor are schedulable by global scheduling. However, due to the difficulty of (randomly) generating such task sets, the experiment does not show the improvement that G-FP-APAE provides over its partitioned and global counterparts for such workloads.

*Experiment 2 (G-FP-APAh vs. G-FP-APAE):* The objective of this experiment is to understand if the performance of the heuristic-based APA schedulability analysis G-FP-APAh is comparable to the exhaustive test G-FP-APAE. We used a similar experimental setup as in the first experiment, but applied both the G-FP-APAh and G-FP-APAE tests. The results in Figure 4 show that G-FP-APAh performs almost as well as to G-FP-APAE, i.e., the curves vary only slightly. This validates the efficiency of the used heuristic. Note that processor affinities were generated randomly in this experiment, which explains the overall lower schedulability compared to Experiment 1.

The experimental results demonstrate that the proposed analysis—reduction to global subproblems—is indeed effective. Further comparisons of APA scheduling with other scheduling techniques and other affinity mask assignment heuristics will certainly be interesting; however, such studies are beyond the scope of this paper and remain the subject of future work.

## VI. CONCLUSION AND OPEN QUESTIONS

In this paper, we investigated the schedulability analysis of real-time task sets with APAs. While processor affinities have been studied and used by application developers for providing isolation and average-case enhancements, this work is the first of its kind that explores APAs from a schedulability perspective. We showed that APA-based JLFP scheduling strictly dominates global, clustered, and partitioned JLFP scheduling. The primary contribution of this paper, however, is our schedulability analysis for APA scheduling, which is simple, efficient, and reuses the extensive body of results for global scheduling already available.<sup>13</sup> In summary, the paper establishes that arbitrary processor affinities are useful and can be analyzed formally. Therefore, we hope to stir further research into the design of improved analysis techniques for APA scheduling and stronger models with more flexible migration strategies.

Since this paper is an initial step in the development of real-time scheduling theory for APA scheduling, there is abundant room for future work. First, a key question: is designing schedulability analysis for APA scheduling based on global schedulability analysis the best we can do? The results from our experiments are inconclusive w.r.t. this question. To answer this question, we need a feasibility test or any other schedulability test for APA scheduling that could be used as a benchmark. However, the major source of pessimism seems to be the computation of interference for affinities with high degree of overlaps, i.e., the interference computation assumes that the processors in the overlapping region have to service the entire demand of all the tasks that can be scheduled on

<sup>13</sup>**Correction:** some, but not all, results for global scheduling can be reused. Compatibility with the proposed reduction-based analysis must be established individually for each analysis prior to reuse; see Appendices A and B.

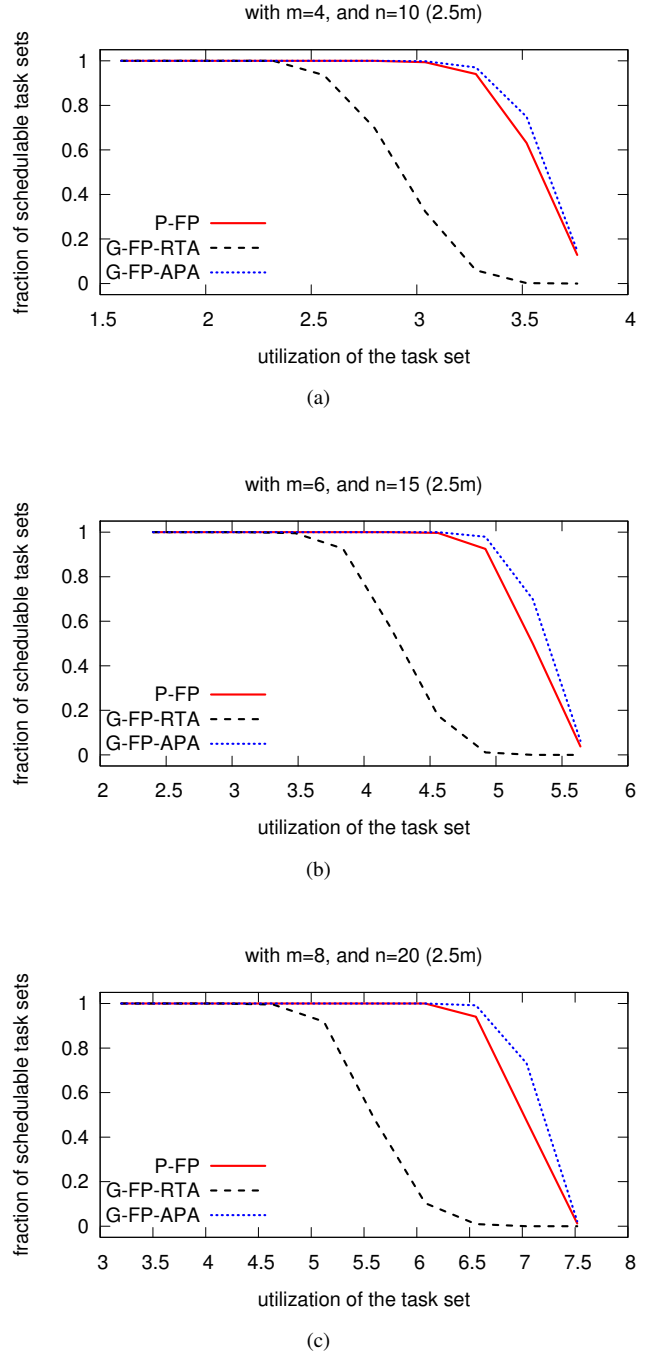


Fig. 3. The comparison of APA scheduling (G-FP-APA) versus global (G-FP-RTA) and partitioned (P-FP) scheduling.

those processors. This assumption is in some cases certainly inaccurate because apart from the overlapping regions, an interfering task can also be scheduled on other processors in its processor affinity. Therefore, the development of new schedulability analysis for APA scheduling from first principles would provide a useful benchmark for future improvements.

However, the assumption that tasks with overlapping affinities interfere with their entire workload is difficult to avoid in the context of *worst-case* analysis since, according to Linux’s

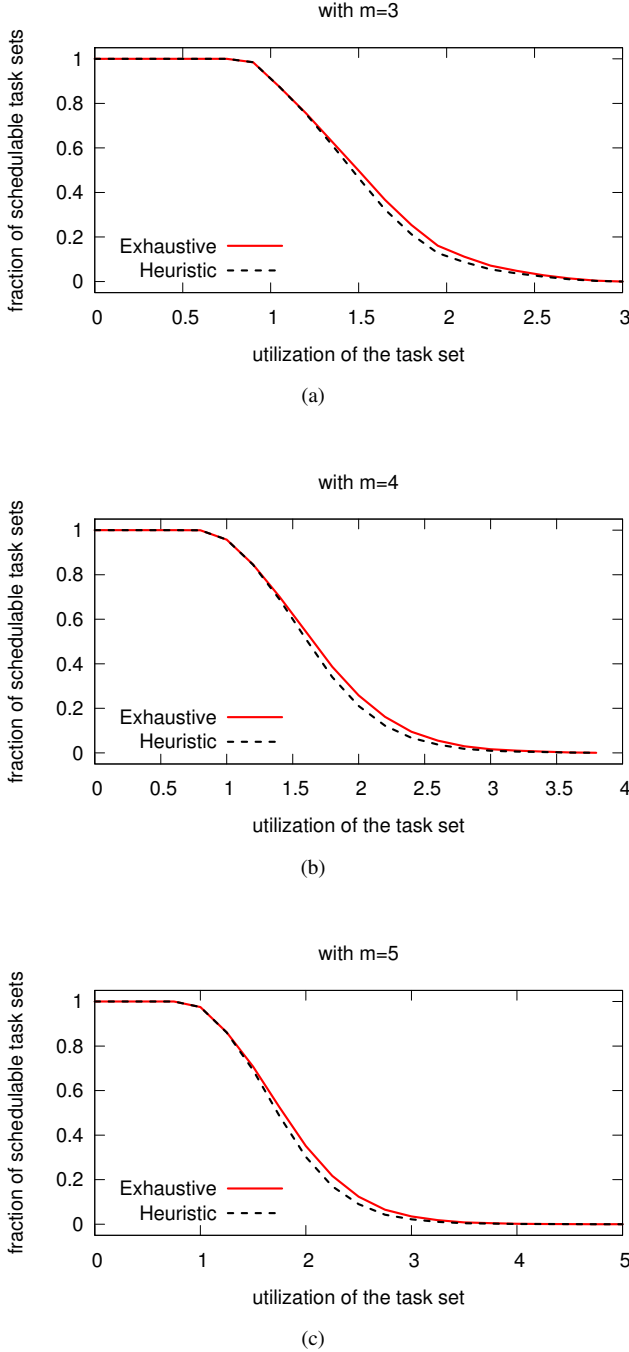


Fig. 4. The comparison of heuristic based G-FP-RTA algorithm w.r.t. the exhaustive G-FP-RTA

push-pull semantics, a higher-priority task interfering with a lower-priority task does not voluntarily yield the processor, even if it could migrate to an otherwise idle processor. That is, the Linux scheduler does not enact a migration if a *scheduled* higher-priority task  $T_h$  (executing on a processor  $\Pi_x$ ) may “altruistically” give up its processor, in lieu of another processor  $\Pi_y$  (also in  $\alpha_h$ ), so that a lower-priority task  $T_l$  can be scheduled on  $\Pi_x$  (effectively *shifting*  $T_h$  from  $\Pi_x$  to  $\Pi_y$ ).

For example, consider a simple task set with implicit

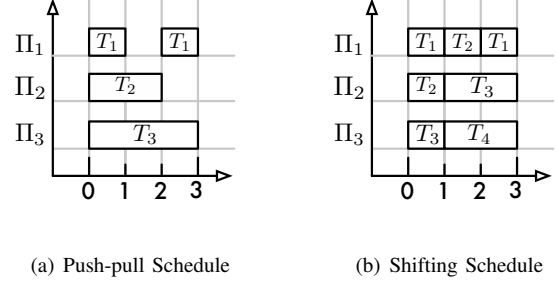


Fig. 5. The figures show the initial schedules of the task set  $T_1(1, 2)$ ,  $T_2(2, 3)$ ,  $T_3(3, 4)$ , and  $T_4(4, 12)$  having processor affinities  $\{\Pi_1\}$ ,  $\{\Pi_1, \Pi_2\}$ ,  $\{\Pi_2, \Pi_3\}$ , and  $\{\Pi_3\}$  respectively, when scheduled by (a) Linux’s push-pull scheduler and (b) a hypothetical shifting-migrations-based scheduler.

deadlines, which consists of tasks  $T_1(1, 2)$ ,  $T_2(2, 3)$ ,  $T_3(3, 4)$ , and  $T_4(4, 12)$  having processor affinities  $\{\Pi_1\}$ ,  $\{\Pi_1, \Pi_2\}$ ,  $\{\Pi_2, \Pi_3\}$ , and  $\{\Pi_3\}$  respectively. Figure 5 illustrates the initial portion of two possible schedules for this task set. Figure 5(a) shows a schedule assuming Linux’s push-pull semantics. In Figure 5(b), at time 1,  $T_2$  shifts from  $\Pi_2$  to  $\Pi_1$  and  $T_3$  shifts from  $\Pi_3$  to  $\Pi_2$  to enable scheduling of  $T_4$ . While in the push-pull schedule,  $T_4$ ’s job finishes its execution at time 12, in the second schedule, it finishes its execution at time 4. Therefore, we conjecture that a stronger migration rule like shifting will allow processor affinities to provide significantly higher schedulability, and so we seek to explore the design space of shifting-based scheduling algorithms in future work.

Finally, APAs do not place any restrictions on *when* migrations can take place. Another obvious generalization of the studied problem would be to interpret each  $\alpha_i$  as function of time (similar to priorities), which could be used to generalize many semi-partitioned schedulers, and other hybrid schedulers, in the literature. There is also a significant room for improvements by exploring the problem of finding optimal processor affinities and optimal priority assignments.

**Acknowledgement.** We thank Sanjoy Baruah for pointing out the connection to the scheduling of unrelated heterogeneous multiprocessors and restricted identical machines, and Geoffrey Nelissen for discovering errors in the original version of this paper (as documented in Appendix A).

## REFERENCES

- [1] J. Anderson, V. Bud, and U. Devi, “An EDF-based scheduling algorithm for multiprocessor soft real-time systems,” in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, ser. ECRTS ’05. IEEE Computer Society, 2005, pp. 199–208.
- [2] B. Andersson, G. Raravi, and K. Bletsas, “Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors,” in *Proceedings of the 31st Real-Time Systems Symposium*, 2010, pp. 239–248.
- [3] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, “Applying new scheduling theory to static priority pre-emptive scheduling,” *Software Eng. J.*, vol. 8, no. 5, 1993.
- [4] N. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, “Hard real-time scheduling: The deadline-monotonic approach,” in *Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software*, 1991, pp. 133–137.
- [5] B. Bado, L. George, P. Courbin, and J. Goossens, “A semi-partitioned approach for parallel real-time scheduling,” in *20th*

- International Conference on Real-Time and Network Systems*. ACM, 2012, pp. 151–160.
- [6] T. Baker and S. Baruah, “Schedulability analysis of multiprocessor sporadic task systems,” in *Handbook of Real-time and Embedded Systems*. CRC Press, 2007.
- [7] S. Baruah, “Techniques for multiprocessor global schedulability analysis,” in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, 2007, pp. 119–128.
- [8] —, “Partitioning real-time tasks among heterogeneous multiprocessors,” in *Proceedings of the International Conference on Parallel Processing*, 2004, pp. 467–474.
- [9] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, “Proportionate progress: A notion of fairness in resource allocation,” *Algorithmica*, vol. 15, pp. 600–625, 1996.
- [10] M. Bertogna and M. Cirinei, “Response-time analysis for globally scheduled symmetric multiprocessor platforms,” in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, 2007, pp. 149–160.
- [11] M. Bertogna, M. Cirinei, and G. Lipari, “Improved schedulability analysis of EDF on multiprocessor platforms,” in *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*. IEEE, 2005, pp. 209–218.
- [12] B. Brandenburg, “Scheduling and locking in multiprocessor real-time operating systems,” Ph.D. dissertation, The University of North Carolina at Chapel Hill, 2011.
- [13] A. Burns, R. Davis, P. Wang, and F. Zhang, “Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme,” *Real-Time Systems*, vol. 48, pp. 3–33, 2012.
- [14] J. Calandrino, J. Anderson, and D. Baumberger, “A hybrid real-time scheduling approach for large-scale multicore platforms,” in *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, 2007, pp. 247–258.
- [15] R. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, 2011.
- [16] —, “Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, 2011.
- [17] M. Dertouzos and A. Mok, “Multiprocessor online scheduling of hard-real-time tasks,” *Software Engineering, IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1497–1506, 1989.
- [18] F. Dorin, P. Yomsi, J. Goossens, and P. Richard, “Semi-partitioned hard real-time scheduling with restricted migrations upon identical multiprocessor platforms,” *CoRR*, vol. abs/1006.2637, 2010.
- [19] P. Emberson, R. Stafford, and R. Davis, “Techniques for the synthesis of multiprocessor tasksets,” *1st Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2010.
- [20] N. Fisher, J. Goossens, and S. Baruah, “Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible,” *Real-Time Systems*, vol. 45, no. 1-2, pp. 26–71, 2010.
- [21] A. Foong, J. Fung, and D. Newell, “An in-depth analysis of the impact of processor affinity on network performance,” in *Proceedings of the 12th IEEE International Conference on Networks*, vol. 1, 2004, pp. 244–250.
- [22] S. Funk, “EDF scheduling on heterogeneous multiprocessors,” Ph.D. dissertation, The University of North Carolina at Chapel Hill, 2004.
- [23] J. Gálvez, P. Ruiz, and A. Skarmeta, “Heuristics for scheduling on restricted identical machines,” University of Murcia, Spain, Tech. Rep., 2010.
- [24] A. Gujarati, F. Cerqueira, and B. Brandenburg, “Schedulability analysis of the Linux push and pull scheduler with arbitrary processor affinities,” in *Proceedings of the 25th Euromicro Conference on Real-Time Systems*, 2013, pp. 69–79.
- [25] S. Kato, N. Yamasaki, and Y. Ishikawa, “Semi-partitioned scheduling of sporadic task systems on multiprocessors,” in *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, 2009, pp. 249–258.
- [26] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, “Deadline scheduling in the linux kernel,” *Software: Practice and Experience*, 2015. [Online]. Available: <http://dx.doi.org/10.1002/spe.2335>
- [27] J. Leung and C.-L. Li, “Scheduling with processing set restrictions: A survey,” *International Journal of Production Economics*, vol. 116, no. 2, pp. 251–262, 2008.
- [28] J. Leung and J. Whitehead, “On the complexity of fixed-priority scheduling of periodic, real-time tasks,” *Performance evaluation*, vol. 2, no. 4, pp. 237–250, 1982.
- [29] C. Liu and J. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.
- [30] A. Mok, “Fundamental design problems of distributed systems for the hard-real-time environment,” Massachusetts Institute of Technology, Tech. Rep., 1983.

## A. ERRATUM — GENERALITY OF REDUCTION

The original version of this paper [24], presented at the 25<sup>th</sup> Euromicro Conference on Real-Time Systems, overstated the generality of the reduction-based analysis. As kindly brought to our attention by Geoffrey Nelissen, the proposed reduction method is in fact not compatible with *all* schedulability tests for global scheduling, and in particular not with exact (i.e., necessary and sufficient) schedulability tests. To this end, the paper has been revised to clearly identify the incorrectly overgeneralized claims, and to establish the correctness of the approach for specific global schedulability tests. In particular, the following changes have been made.

- 1) In Appendix A.1, we document and explain the counterexample to Lemma 2 provided by Geoffrey Nelissen that illustrates the limits of the reduction.
- 2) In the process of revising the paper, we became aware that Lemma 4, which is stated in terms that suggest that it applies generally to all JLFP policies, does in fact not apply to certain JLFP policies such as EDF. In Appendix A.2, we document a counterexample.
- 3) We revised Sec. IV to mark Lemmas 2 and 4, and all dependent claims, as incorrect.
- 4) In Appendix B, we show Bertogna and Cirinei’s response-time analyses for FP and EDF scheduling (both with and without slack updates) [10] to be compatible with the proposed reduction approach.

Finally, we note that the experiments reported on in Sec. V are not affected by this erratum because the employed global schedulability test — Bertogna and Cirinei’s response-time analysis for FP scheduling [10] — is in fact compatible, as argued in Appendix B.2.

#### A.1 Counterexample — Not All Global Schedulability Tests are Compatible with the Proposed Reduction Technique

In this appendix, we summarize the counterexample provided by Geoffrey Nelissen, which contradicts the following overly general claim made in the original paper.

**Refuted Claim 1 (Lemma 2):** “If a task  $T_i \in \text{tasks}(\alpha_i)$  is schedulable when the reduced task set  $\text{tasks}(\alpha_i)$  is globally scheduled on the reduced processor platform  $\alpha_i$  using a JLFP policy  $A$ , then  $T_i$  is also schedulable under APA scheduling of  $\tau$  on the processor platform  $\pi$  using the same JLFP policy  $A$ .”

Differently from what has been implicitly stated, the reduction technique does not work with every global schedulability analysis, as demonstrated by the following examples.

##### Example 3 (Nelissen’s example – periodic tasks):

Consider task set  $\tau^{per} = \{T_1 = (2, 10), T_2 = (2, 10), T_3 = (4, 10), T_4 = (7, 10)\}$ . If job arrivals are assumed to be periodic, we can derive an exact schedulability analysis that uses the hyperperiod as a feasibility interval. From the simulation of the schedule shown in Figure 6 (assuming FP

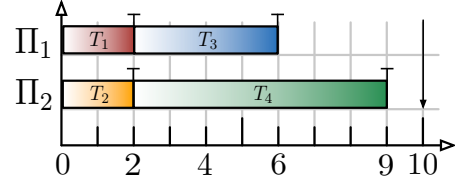


Fig. 6. FP schedule of task set  $\tau^{per}$  under global scheduling. Note that task  $T_3$  does not interfere with  $T_4$ , which completes at time 9 and thus meets its deadline at time 10.

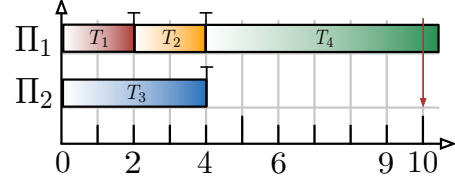


Fig. 7. FP schedule of task set  $\tau^{per}$  under APA scheduling if tasks  $T_1$  and  $T_2$  are restricted to processor  $\Pi_1$ . Due to the resulting loss of parallelism, the interference of  $T_3$  on  $T_4$  increases by 4 time units (w.r.t. Figure 6), which causes  $T_4$  to miss its deadline at time 10.

scheduling), it follows that task set  $\tau^{per}$  is schedulable under global scheduling.

Next, consider the same task set under APA scheduling, where tasks  $T_1$  and  $T_2$  are restricted to processor  $\Pi_1$  (i.e.,  $\alpha_1 = \alpha_2 = \{\Pi_1\}$ ), and  $T_3$  is restricted to processor  $\Pi_2$  (i.e.,  $\alpha_3 = \{\Pi_2\}$ ). Consider the schedulability of the lowest-priority task  $T_4$ , which can execute on both processors ( $\alpha_4 = \{\Pi_1, \Pi_2\}$ ). In the corresponding schedule, which is shown in Figure 7, a lack of parallelism causes task  $T_4$  to miss its deadline at time 10. Therefore, when an exact schedulability analysis is employed, the fact that task  $T_4$  is schedulable under global scheduling (Figure 6) does not imply that it is schedulable under APA scheduling (Figure 7).

Example 3 shows that Claim 1 does not hold for periodic tasks. Furthermore, while this observation by itself does not necessarily extend to sporadic tasks, it is also possible to derive a similar counterexample assuming sporadic tasks, as shown next. As in Example 3, the key observation is that, by restricting processor affinities of higher-priority tasks, interference that coincides under global scheduling is serialized under APA scheduling, leading to a deadline miss in a lower-priority task.

##### Example 4 (Nelissen’s example – sporadic tasks):

Consider a sporadic task set  $\tau^{spo} = \{T_1 = (1, 2), T_2 = (1, 3), T_3 = (5, 1000), T_4 = (1, 5)\}$  to be scheduled under global FP scheduling. We first argue that task  $T_4$  is schedulable for any job arrival sequence, by considering two cases for the arrival times of tasks  $T_1$  and  $T_2$ . Let  $J$  denote an arbitrary job of  $T_4$ .

Case 1: Tasks  $T_1$  and  $T_2$  do not release a job simultaneously while  $J$  is pending. Consider the time window of length four starting with the arrival of  $J$  and ending at its absolute deadline. Even if tasks  $T_1$  and  $T_2$  release jobs as fast as possible (limited by their periods), there can be no more than three jobs of  $T_1$  and  $T_2$  in such a time window (as their job arrival times never coincide), which limits the inference incurred by  $J$  due to  $T_1$  and  $T_2$  to at most three time units. Moreover, even if task  $T_3$  occupies a processor throughout the interval, it causes at most



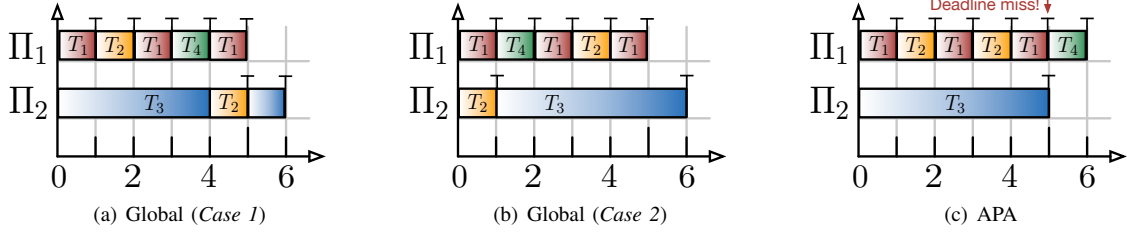


Fig. 8. FP schedules of task set  $\tau^{spo}$  under global and APA scheduling. Insets (a) and (b) correspond, respectively, to Case 1 (tasks  $T_1$  and  $T_2$  never release a job simultaneously) and Case 2 (tasks  $T_1$  and  $T_2$  release jobs at the same time at least once) in Example 4. For simplicity, only a single job of task  $T_4$  is shown. Since the interference incurred by  $T_4$ 's job in either case does not exceed four time units,  $T_4$  is schedulable under global scheduling. Inset (c) corresponds to the schedule of the same task set under APA scheduling, assuming a synchronous task set release with periodic arrival times and restricted processor affinities  $\alpha_1 = \alpha_2 = \{\Pi_1\}$ ,  $\alpha_3 = \{\Pi_2\}$  and  $\alpha_4 = \{\Pi_1, \Pi_2\}$ . Due to the loss of parallelism caused by affinity restrictions, task  $T_4$  incurs five times units of interference and misses a deadline.

four time units of interference, for a total of at most seven time units of interference due to  $T_1$ ,  $T_2$ , and  $T_3$ . Thus, there exists at least one time at which one of the two processors is available for task  $T_4$  to meet its deadline. This worst-case scenario is depicted in Figure 8(a).

Case 2: There exists at least one time  $t$  at which both tasks  $T_1$  and  $T_2$  release jobs while  $J$  is pending. In that case, the jobs of tasks  $T_1$  and  $T_2$  released at time  $t$  execute in parallel and task  $T_4$  is only interfered with for one time unit (instead of two as in Case 1). Thus, as shown in Figure 8(b), the total interference incurred by task  $J$  is not larger than in Case 1, where job arrivals of  $T_1$  and  $T_2$  do not coincide.

In both cases, the response time of  $J$  is limited to four time units, which implies that  $T_4$  meets all deadlines under global scheduling. However, this is not the case when certain processor affinities are enforced: as in Example 3, if  $\alpha_1 = \alpha_2 = \{\Pi_1\}$ ,  $\alpha_3 = \{\Pi_2\}$  and  $\alpha_4 = \{\Pi_1, \Pi_2\}$ , then task  $T_4$  misses a deadline, as shown in Figure 8(c). Thus, the fact that task  $T_4$  is schedulable under global scheduling (Figures 8(a) and 8(b)) does not imply that it is schedulable under APA scheduling (Figure 8(c)), which contradicts Claim 1.

Nelissen's counterexamples expose scenarios in which the proposed reduction fails when combined with an exact schedulability analysis for global scheduling. Although we compare schedules with the same set of interfering tasks, some properties of the task set (e.g., the lowest upper bound on task interference) may change when processor affinities are introduced. Further, while certain sufficient (but not necessary) global schedulability tests can be applied directly because the pessimism inherent in these tests masks the difference in interference between global and APA scheduling, there is no guarantee that all sufficient (but not necessary) tests are compatible. Thus, each test must be shown to be compatible before it may be applied in the proposed reduction technique.

In Appendix B, we show three response-time analyses proposed by Bertogna and Cirinei [10] to be compatible.

#### A.2 Counterexample — Enlarging a Task's Affinity Does Not Necessarily Preserve Schedulability under EDF

In the process of reviewing Sec. IV, we observed that Lemma 4, reproduced below, was stated in incorrectly over-generalized terms, too.

**Refuted Claim 2 (Lemma 4):** “If a task  $T_i \in \tau$  is schedulable under APA scheduling with the processor affinity  $\alpha'_i \subset \alpha_i$  and task set  $\tau$ , then  $T_i$  is also schedulable under APA scheduling with the affinity  $\alpha_i$  and task set  $\tau$ .”

The lemma in this form does not hold under all JLFP policies. In the following, we document a counterexample to the claim.

According to the APA scheduling invariant, and also as implemented by Linux's APA scheduler, a task  $T_i$  can (non-deterministically) start execution on any processor in its affinity  $\alpha_i$  and preempt lower-priority jobs, independently of how restrictive the affinities of the lower-priority jobs are. Furthermore, under a JLFP policy such as EDF, a task  $T_i$  can indirectly delay one of its own subsequent jobs by preempting a lower-priority job that represents higher-priority demand w.r.t. a future job of  $T_i$ . That is, preempting a lower-priority job can increase the magnitude of future higher-priority interference.

Hence, for certain JLFP policies such as EDF, enlarging the affinity of a task  $T_i$  from  $\alpha'_i$  to  $\alpha_i$  is not guaranteed to preserve schedulability of  $T_i$  because it can increase the interference on other tasks that execute on processors in  $\alpha_i \setminus \alpha'_i$ , which in turn can transitively delay a subsequent job of  $T_i$ . Such a scenario is demonstrated in the following example.

**Example 5:** Consider a constrained-deadline task set  $\tau'$  consisting of tasks  $T_1 = (1, 5, 10)$ ,  $T_2 = (1, 5, 10)$ ,  $T_3 = (4, 5, 10)$ , and  $T_4 = (8, 13, 13)$ , to be scheduled on a platform  $\pi = \{\Pi_1, \Pi_2\}$  under APA scheduling using EDF. The processor affinities of the tasks in  $\tau$  are given by  $\alpha_1 = \{\Pi_1, \Pi_2\}$ ,  $\alpha_2 = \{\Pi_1, \Pi_2\}$ ,  $\alpha'_3 = \{\Pi_2\}$  and  $\alpha_4 = \{\Pi_1\}$ , respectively. Assume that  $T_3$  is the task to be analyzed and that ties in absolute deadline are broken in favor of lower-indexed tasks.

Figure 9 shows the schedule of  $\tau'$  assuming a synchronous, periodic arrival sequence. As denoted in the figure as  $d_{i,j}$ , the first jobs of  $T_1$ ,  $T_2$ , and  $T_3$  each have an absolute deadline at time 5; the job of  $T_4$  has an absolute deadline at time 13, and the second jobs of  $T_1$ ,  $T_2$ , and  $T_3$  each have an absolute deadline at time 15.

For this particular arrival sequence, and also in the general case, task  $T_3$  never misses any deadline. This follows from the fact that tasks  $T_1$  and  $T_2$  can either be released simultaneously (as depicted), or at different times (not shown in the figure), but



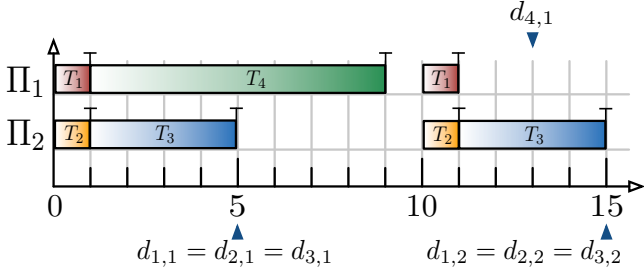


Fig. 9. EDF schedule of task set  $\tau'$  under APA scheduling. With a restricted affinity  $\alpha'_3 = \{\Pi_2\}$ , task  $T_3$  meets its deadline at time 15.

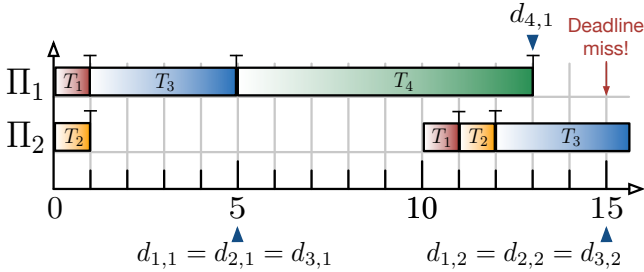


Fig. 10. EDF schedule of task set  $\tau$  under APA scheduling. When task  $T_3$  has a larger affinity  $\alpha_3 = \{\Pi_1, \Pi_2\}$ , it can delay a job of  $T_4$ , which in turn can interfere with the later-released second job of  $T_3$ . As a result,  $T_3$  misses its deadline at time 15.

in both cases there is always enough time for  $T_3$  to complete, since the total delay experienced by any of  $T_3$ 's jobs due to  $T_1$  and  $T_2$  never exceeds one time unit:

- if  $T_1$  and  $T_2$  release jobs simultaneously, they will execute in parallel; and
- if one releases a job later than the other, at least one of them either has an absolute deadline past  $T_3$ 's current job's absolute deadline, or will have completed before  $T_3$ 's next job is released.

Next, consider a task set  $\tau$  that is virtually identical to  $\tau'$ , except that task  $T_3$  has a larger affinity  $\alpha_3 = \{\Pi_1, \Pi_2\}$ . As depicted in Figure 10, if task  $T_3$  is released and starts execution on processor  $\Pi_1$  (as allowed under the APA scheduling invariant), this delays task  $T_4$ , which is not able to execute elsewhere due to its restrictive affinity. Since, under EDF,  $T_4$ 's jobs can interfere with later-released jobs of the other tasks, this additional delay generates backlogged demand that affects the second job of task  $T_3$ , causing a deadline miss at time 15.

This example shows that, even though task  $T_3$  is schedulable assuming a reduced affinity  $\alpha'_3 = \{\Pi_2\}$ , it is not schedulable when assuming a larger affinity  $\alpha_3 = \{\Pi_1, \Pi_2\}$ , which contradicts Claim 2.

Nonetheless, the intuition that it is sufficient to consider only a subset of a task's affinity when analyzing interference is not entirely wrong; Lemma 6 in Appendix B.1 restates the basic idea underlying Claim 2 in more accurate terms.

## B. APA-COMPATIBILITY OF RESPONSE-TIME ANALYSIS

In this appendix, we show three response-time analyses (RTAs) proposed by Bertogna and Cirinei [10] to be compatible with the proposed reduction approach:

- 1) RTA for FP scheduling *without* slack updates, which was used in the experiments reported on in Sec. V,
- 2) RTA for FP scheduling *with* slack updates, and
- 3) RTA for EDF scheduling *with* slack updates (which subsumes RTA for EDF scheduling *without* slack updates).

These RTAs for global scheduling are compatible with APA scheduling in the sense that, if the corresponding equations are instantiated for global-like subproblems (for each task) as described in Sec. IV, then the task set is schedulable under APA scheduling if for each equation a valid solution can be found (i.e., one not exceeding each task's relative deadline). We establish this notion of compatibility in detail in Appendices B.2–B.4 after first introducing required definitions and auxiliary lemmas.

### B.1 Preliminaries

Central to Bertogna and Cirinei's analysis [10] are the notions of “workload” and “interference,” which we briefly recall here in preparation.

- A task  $T_k$ 's *workload* during a given interval  $[t_0, t_0 + \Delta]$  is the maximum total time that jobs of  $T_k$  use any processor during  $[t_0, t_0 + \Delta]$ . Regardless of the employed scheduler, it is possible to upper-bound the maximum total workload of  $T_k$  in any contiguous interval of length  $\Delta$  based solely on  $T_k$ 's period, deadline, and WCET, assuming that  $T_k$  does not miss any deadlines.
- A task  $T_k$ 's execution at a time  $t$  constitutes *interference* for another task  $T_i$  iff  $T_i$  is backlogged at time  $t$  (i.e.,  $T_i$  has a ready job that is not scheduled) and a job of  $T_k$  is scheduled on a processor on which  $T_i$  may execute (i.e., a processor in  $\alpha_i$  under APA scheduling, or any processor in  $\pi$  under global scheduling).

Clearly, a task  $T_k$  causes another task  $T_i$  to incur interference only if  $T_k$ 's current job has higher priority than  $T_i$ 's backlogged job. As the maximum total interference caused by  $T_k$  in a given interval is bounded by its workload (a task must be scheduled to interfere), Bertogna and Cirinei [10] established the following generic workload bound to analyze interference.

**Definition 1 (workload bound [10]):** Consider an arbitrary interval  $[t_0, t_0 + \Delta]$  of length  $\Delta$  and a constrained-deadline task  $T_k$ . If all jobs of  $T_k$  with deadlines before  $t_0 + \Delta$  do not miss their deadline, then  $T_k$ 's maximum workload during  $[t_0, t_0 + \Delta]$  is bounded (under any scheduler) by

$$W_k(\Delta) = n_k(\Delta) \cdot e_k + \min(e_k, \Delta + d_k - e_k - n_k(\Delta) \cdot p_k),$$

where  $n_k(\Delta) = \left\lfloor \frac{\Delta + d_k - e_k}{p_k} \right\rfloor$  upper-bounds the number of jobs that execute for their entire WCET during  $[t_0, t_0 + \Delta]$ .

Bertogna and Cirinei [10] further refined their workload bound by taking into account that a known bound on a task's maximum response time also implies a lower bound on its “slack,” i.e., the difference between the task's maximum response time and its relative deadline. If a task has non-zero slack, there exists a “gap” between any two consecutive job releases during which the task does not have a pending job,

which in turn imposes a bound on the maximum work that the task can carry into a contiguous interval. Bertogna and Cirinei [10] exploited this observation to derive the following slack-aware bound.

**Definition 2** (slack-aware workload bound [10]):

Consider an arbitrary interval  $[t_0, t_0 + \Delta]$  of length  $\Delta$  and a constrained-deadline task  $T_k$ . If the response times of all jobs of  $T_k$  with deadlines before  $t_0 + \Delta$  are bounded by  $R_k$ , where  $R_k \leq d_k$ , then  $T_k$ 's maximum workload during  $[t_0, t_0 + \Delta]$  is bounded (under any scheduler) by

$$W'_k(\Delta) = n'_k(\Delta) \cdot e_k + \min(e_k, \Delta + d_k - e_k - s_k - n'_k(\Delta) \cdot p_k),$$

where  $s_k = d_k - R_k$  is a lower bound on the *slack* of task  $T_k$ , and  $n'_k(\Delta) = \left\lfloor \frac{\Delta + d_k - e_k - s_k}{p_k} \right\rfloor$  upper-bounds the number of jobs that execute for their entire WCET during  $[t_0, t_0 + \Delta]$ .

Definitions 1 and 2 are generic in the sense that they apply to any work-conserving scheduler. Bertogna and Cirinei [10] further established an interference bound specific to EDF that takes into account that not all of a task's workload may constitute interference under EDF due to the dynamic prioritization of jobs.

**Definition 3** (interference bound for EDF [10]): Consider a constrained-deadline task  $T_k$  and an interval  $[t_0, t_0 + d_i]$  of length  $d_i$  that starts with the release of a job  $J$  of another task  $T_i$  and that ends at the absolute deadline of  $J$ . If the response times of all jobs of  $T_k$  with deadlines before  $t_0 + d_i$  are bounded by  $R_k$ , where  $R_k \leq d_k$ , then, under EDF, the maximum interference incurred by  $J$  due to jobs of  $T_k$  is bounded by

$$\mathcal{J}_i^k = DBF_i^k + \min\left(e_k, \max\left(0, d_i - DBF_i^k \cdot \frac{p_k}{e_k} - s_k\right)\right),$$

where  $s_k = d_k - R_k$  is a lower bound on the slack of task  $T_k$ , and where  $DBF_i^k = \left(\left\lfloor \frac{d_i - d_k}{p_k} \right\rfloor + 1\right) \cdot e_k$  is an upper bound on the *demand* of task  $T_k$  in the scheduling window of any job of task  $T_i$  (i.e., the maximum cumulative execution requirement of all jobs of task  $T_k$  that both are released and have a deadline in a contiguous interval of length  $d_i$ ).

Since the individual bounds that comprise  $\mathcal{J}_i^k$  depend only on fixed tasks parameters, the assumption that jobs of  $T_k$  respect a known response-time bound  $R_k$ , and the fact that jobs are prioritized according to EDF,  $\mathcal{J}_i^k$  is valid under both global and APA scheduling (provided a suitable scheduler-specific bound  $R_k$  has been established).

Next, we establish two auxiliary lemmas concerning the total interference incurred by a job under APA scheduling. First, we observe that a lower bound on total interference also implies a lower bound on interference due to tasks that may execute on a subset of the processors.

**Lemma 6:** Let  $J$  denote a job of task  $T_i \in \tau$ , and let  $t_r$  denote  $J$ 's release time. Consider an interval  $[t_r, t_r + \Delta]$  of length  $\Delta$ , and let  $x_k$  denote the interference incurred by  $J$  due to task  $T_k \in \tau$  during  $[t_r, t_r + \Delta]$ , i.e., the total time that jobs

of  $T_k$  execute on processors in  $\alpha_i$  during  $[t_r, t_r + \Delta]$  while  $J$  is pending and not scheduled. Let  $\alpha'_i \subseteq \alpha_i$  denote an arbitrary non-empty subset of processors on which  $T_i$  may be scheduled. Under APA scheduling with any JLFP policy, if

$$\sum_{T_k \in \tau} x_k \geq |\alpha_i| \cdot (\Delta - e_i + 1),$$

then also

$$\sum_{T_k \in \text{tasks}(\alpha'_i)} x_k \geq |\alpha'_i| \cdot (\Delta - e_i + 1).$$

*Proof:* By definition of each  $x_k$ ,  $J$  incurs interference for exactly  $\frac{\sum_{T_k \in \tau} x_k}{|\alpha_i|} \geq \Delta - e_i + 1$  time units during  $[t_r, t_r + \Delta]$ . A task  $T_k$  causes  $T_i$  to incur interference only when all processors in  $\alpha_i$  are occupied by tasks other than  $T_i$ . Thus, each of the processors in  $\alpha'_i$  must be busy executing jobs other than  $J$  while  $J$  is pending for at least  $\Delta - e_i + 1$  time units, for a total of at least  $|\alpha'_i| \cdot (\Delta - e_i + 1)$  time units consumed by higher-priority jobs across all processors in  $\alpha'_i$  while  $J$  is pending. Since only tasks in  $\text{tasks}(\alpha'_i) = \{T_k \mid T_k \in \tau \wedge \alpha_k \cap \alpha'_i \neq \emptyset\}$  can execute on processors in  $\alpha'_i$ , it follows that  $\sum_{\substack{T_k \in \tau \\ \alpha_k \cap \alpha'_i \neq \emptyset}} x_k = \sum_{T_k \in \text{tasks}(\alpha'_i)} x_k \geq |\alpha'_i| \cdot (\Delta - e_i + 1)$ . ■

Finally, Lemma 7, which is modeled after Lemma 4 in [11], establishes that each task's contribution to the total interference can be safely capped.

**Lemma 7:** Let  $J$  denote a job of task  $T_i \in \tau$ , and let  $t_r$  denote  $J$ 's release time. Consider an interval  $[t_r, t_r + \Delta]$  of length  $\Delta$ , and let  $x_k$  denote the interference incurred by  $J$  due to task  $T_k \in \tau$  during  $[t_r, t_r + \Delta]$ , i.e., the total time that jobs of  $T_k$  execute on processors in  $\alpha_i$  during  $[t_r, t_r + \Delta]$  while  $J$  is pending and not scheduled. Let  $\alpha'_i \subseteq \alpha_i$  denote an arbitrary non-empty subset of processors on which  $T_i$  may be scheduled. Under APA scheduling with any JLFP policy, if

$$\sum_{T_k \in \text{tasks}(\alpha'_i)} x_k \geq |\alpha'_i| \cdot (\Delta - e_i + 1),$$

then also

$$\sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, \Delta - e_i + 1) \geq |\alpha'_i| \cdot (\Delta - e_i + 1).$$

*Proof:* Let  $A \subseteq \text{tasks}(\alpha'_i)$  denote the set of tasks in  $\text{tasks}(\alpha'_i)$  that cause  $J$  more than  $\Delta - e_i + 1$  interference during  $[t_r, t_r + \Delta]$ , i.e.,

$$A \triangleq \{T_j \mid T_j \in \text{tasks}(\alpha'_i) \wedge x_j > \Delta - e_i + 1\},$$

and let  $B \triangleq \text{tasks}(\alpha'_i) \setminus A$ . Then:

$$\begin{aligned} & \sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, \Delta - e_i + 1) \\ &= \sum_{T_k \in A} (\Delta - e_i + 1) + \sum_{T_k \in B} x_k \\ &= |A| \cdot (\Delta - e_i + 1) + \sum_{T_k \in B} x_k. \end{aligned}$$

We consider three cases based on the cardinality of  $A$ .

*Case 1:* If  $|A| = 0$ , then trivially  $B = \text{tasks}(\alpha'_i)$ , and thus  $\sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, \Delta - e_i + 1) = |A| \cdot (\Delta - e_i + 1) + \sum_{T_k \in B} x_k = \sum_{T_k \in \text{tasks}(\alpha'_i)} x_k \geq |\alpha'_i| \cdot (\Delta - e_i + 1)$ .

*Case 2:* If  $|A| \geq |\alpha'_i|$ , then tasks in  $B$  are irrelevant and trivially  $\sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, \Delta - e_i + 1) \geq |A| \cdot (\Delta - e_i + 1) \geq |\alpha'_i| \cdot (\Delta - e_i + 1)$ .

*Case 3:* If  $0 < |A| < |\alpha'_i|$ , then the interference due to tasks in both  $A$  and  $B$  needs to be considered. Because a task causes interference only when all processors in  $\alpha'_i$  are taken up by jobs other than  $J$ , any task's interference implies a lower bound on the interference caused by other tasks. In particular, since  $0 < |A| < |\alpha'_i|$ , whenever one of the tasks in  $A$  causes interference, at least  $|\alpha'_i| - |A|$  of the cores in  $\alpha'_i$  are taken up by tasks in  $B$ , and thus

$$\begin{aligned} \sum_{T_k \in B} x_k &\geq (|\alpha'_i| - |A|) \cdot \min_{T_j \in A} \{x_j\} \\ &\geq (|\alpha'_i| - |A|) \cdot (\Delta - e_i + 1). \end{aligned}$$

Then:

$$\begin{aligned} &\sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, \Delta - e_i + 1) \\ &= |A| \cdot (\Delta - e_i + 1) + \sum_{T_k \in B} x_k \\ &\geq |A| \cdot (\Delta - e_i + 1) + (|\alpha'_i| - |A|) \cdot (\Delta - e_i + 1) \\ &= |\alpha'_i| \cdot (\Delta - e_i + 1). \end{aligned}$$

## B.2 FP RTA without Slack Updates

In this appendix, we establish that Bertogna and Cirinei's response-time analysis for FP scheduling without slack updates [10], which was used in the experiments reported on in Sec. V, can be safely applied under APA scheduling using the reduction approach described in Sec. IV. We begin with a review of the original analysis for global scheduling.

*Review:* Assume sporadic tasks with constrained deadlines. Let  $I_i^k(t)$  denote an upper bound on the interference of a higher-priority task  $T_k$  on task  $T_i$  in an interval of length  $t$  that starts with the release of a job of  $T_i$ . Based on the workload bound (Definition 1),  $I_i^k(t)$  is defined as follows:

$$I_i^k(t) = \min(W_k(t), t - e_i + 1). \quad (12)$$

For notational convenience, we set  $I_i^k(t) = 0$  if task  $T_k$ 's priority is lower than task  $T_i$ 's priority or if  $i = k$ .

Based on the per-task interference bound  $I_i^k(t)$  and task priorities, an upper bound on the maximum response time of task  $T_i$  under global FP scheduling is given by the smallest positive solution to the recurrence

$$R_i^G = e_i + \frac{1}{m} \cdot \sum_{T_k \in \tau} I_i^k(R_i^G), \quad (13)$$

assuming no deadline is missed by any higher-priority task in  $\tau$ . If Equation 13 does not converge to a fixed point less than or

equal to  $d_i$ , then task  $T_i$  is assumed to be not schedulable under global FP scheduling and the upper bound on its maximum response time remains undefined.

*Proof of compatibility:* We next show that, under analogous preconditions and for a given subset of processors  $\alpha'_i \subseteq \alpha_i$ , a valid solution to Equation 13 instantiated for the global-like subproblem induced by  $\alpha'_i$  — with  $m$  replaced by  $|\alpha'_i|$  and  $\tau$  replaced by  $\text{tasks}(\alpha'_i)$  — implies schedulability of  $T_i$  under APA scheduling.

**Lemma 8:** *Let  $\tau$  denote a task set to be scheduled under FP APA scheduling on a processor platform  $\pi$ . For any  $T_i \in \tau$ , if no deadline is missed by any higher-priority task in  $\tau$  (under FP APA scheduling on platform  $\pi$ ), and if there exists a non-empty set of processors  $\alpha'_i \subseteq \alpha_i$  such that Equation 14*

$$R_i^{apa} = e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} I_i^k(R_i^{apa}) \quad (14)$$

*has a least positive solution  $R_i^{apa} \leq d_i$ , then  $R_i^{apa}$  upper-bounds task  $T_i$ 's response time in any schedule of  $\tau$  under FP APA scheduling.*

*Proof:* By contradiction. Suppose that Equation 14 has a solution  $R_i^{apa} \leq d_i$  and no deadline is missed by any higher-priority task in  $\text{tasks}(\alpha'_i)$ , but there exists a schedule  $S$  of  $\tau$  under APA scheduling in which a job of  $T_i$  has a response time greater than  $R_i^{apa}$ , i.e., in which a job of  $T_i$  remains pending for more than  $R_i^{apa}$  time units.

Let  $J$  denote the first job of  $T_i$  to remain pending for more than  $R_i^{apa}$  time units in  $S$  and let  $t_r$  denote  $J$ 's release time. Let  $X \subseteq [t_r, t_r + R_i^{apa})$  denote all times at which  $J$  is interfered with before time  $t_r + R_i^{apa}$ , i.e., all times at which  $J$  is not scheduled during  $[t_r, t_r + R_i^{apa})$ . Since  $J$  does not complete by time  $t_r + R_i^{apa}$ , it is scheduled for strictly less than  $e_i$  time units during  $[t_r, t_r + R_i^{apa})$ , and therefore  $|X| \geq t_r + R_i^{apa} - t_r - (e_i - 1) = R_i^{apa} - e_i + 1$ .

Let  $x_k$  denote the interference incurred by  $J$  due to task  $T_k$  before time  $t_r + R_i^{apa}$ , i.e., the total time that jobs of  $T_k$  execute on processors in  $\alpha_i$  during  $[t_r, t_r + R_i^{apa})$  while  $J$  is not scheduled.

Since under FP scheduling only higher-priority tasks cause interference, and because no deadline is missed by any higher-priority task, we observe that, according to Definition 1,

$$x_k \leq W_k(R_i^{apa}) \quad (15)$$

since a task must be scheduled to interfere.

Since all processors in  $\alpha_i$  are busy executing jobs of higher-priority tasks when  $J$  is not scheduled, we have

$$\sum_{T_k \in \tau} x_k = |X| \cdot |\alpha_i| \geq |\alpha_i| \cdot (R_i^{apa} - e_i + 1),$$

and therefore, by Lemmas 6 and 7, also

$$\sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, R_i^{apa} - e_i + 1) \geq |\alpha'_i| \cdot (R_i^{apa} - e_i + 1),$$

or, rearranged and strengthened to a strict inequality,

$$e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, R_i^{apa} - e_i + 1) > R_i^{apa}.$$

It follows, by the definition of  $R_i^{apa}$  (Equation 14), that

$$\begin{aligned} e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, R_i^{apa} - e_i + 1) \\ > e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} I_i^k(R_i^{apa}), \end{aligned}$$

or, simplified,

$$\sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, R_i^{apa} - e_i + 1) > \sum_{T_k \in \text{tasks}(\alpha'_i)} I_i^k(R_i^{apa}).$$

This implies that there exists a task  $T_k \in \text{tasks}(\alpha'_i)$  such that

$$\min(x_k, R_i^{apa} - e_i + 1) > I_i^k(R_i^{apa}).$$

Since lower-priority tasks cannot cause interference under FP scheduling, and since  $T_i$  does not interfere with itself (recall that  $J$  is the first job with a response time greater than  $R_i^{apa} \leq d_i$  and the restriction to constrained deadlines), it follows that  $x_k > 0$  is possible only if  $T_k$  is a higher-priority task. Therefore, we have  $I_i^k(t) = \min(W_k(t), t - e_i + 1)$ , and thus

$$\min(x_k, R_i^{apa} - e_i + 1) > \min(W_k(R_i^{apa}), R_i^{apa} - e_i + 1).$$

However, this implies  $x_k > W_k(R_i^{apa})$ , which by Inequality 15 is impossible. Contradiction. ■

Schedulability of the entire task set may be established by applying Lemma 8 to each task in  $\tau$  in order of decreasing priority. Most importantly, Lemma 8 shows that it is safe to employ Bertogna and Cirinei's RTA for FP scheduling without slack updates [10] as the underlying schedulability test in the reduction approaches proposed in Lemmas 3 and 5 and Algorithm 1. The results reported on in Sec. V are thus not affected by this erratum.

### B.3 FP RTA with Slack Updates

In this appendix, we argue that considering slack in the basic RTA (Appendix B.2) preserves compatibility with the proposed reduction-based approaches. First, we review the underlying slack-aware response-time bound [10].

*Review:* Based on the slack-aware workload bound  $W'_k(t)$  for constrained-deadline tasks (Definition 2), Bertogna and Cirinei [10] defined the following refined interference bound:

$$I_i^{k'}(t) = \min(W'_k(t), t - e_i + 1). \quad (16)$$

As before, we set  $I_i^{k'}(t) = 0$  if task  $T_k$ 's priority is lower than task  $T_i$ 's priority or if  $i = k$ .

The response-time bound  $R_i^G$  for task  $T_i$  under global FP scheduling is then simply defined as the least positive solution of the equation

$$R_i^G = e_i + \frac{1}{m} \cdot \sum_{T_k \in \tau} I_i^{k'}(R_i^G) \quad (17)$$

that is not greater than deadline  $d_i$ , assuming that for each higher-priority task  $T_h$  a response-time bound not exceeding  $d_h$  is known. Each response-time bound  $R_i^G$  is computed in order of decreasing task priority, starting with the highest-priority task  $T_1$ , for which  $R_1^G = e_1$ .

*Proof of compatibility:* We argue that, under analogous preconditions and for a given subset of processors  $\alpha'_i \subseteq \alpha_i$ , a valid solution to Equation 17 instantiated for the global-like subproblem induced by  $\alpha'_i$  — with  $m$  replaced by  $|\alpha'_i|$  and  $\tau$  replaced by  $\text{tasks}(\alpha'_i)$  — implies schedulability of  $T_i$  under APA scheduling.

**Lemma 9:** Let  $\tau$  denote a task set to be scheduled under FP APA scheduling on a processor platform  $\pi$ . For any task  $T_i \in \tau$ , if an upper bound on the maximum response-time  $R_k^{apa} \leq d_k$  is known for each higher-priority task  $T_k \in \tau$  (under FP APA scheduling on platform  $\pi$ ), and if there exists a non-empty set of processors  $\alpha'_i \subseteq \alpha_i$  such that Equation 18

$$R_i^{apa} = e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} I_i^{k'}(R_i^{apa}) \quad (18)$$

has a least positive solution  $R_i^{apa} \leq d_i$ , then  $R_i^{apa}$  upper-bounds task  $T_i$ 's response time in any schedule of  $\tau$  under FP APA scheduling.

*Proof:* The lemma follows virtually identically to Lemma 8: suppose a job of  $T_i$  exceeds the claimed response-time bound  $R_i^{apa}$ , and define  $x_k$  as in the proof of Lemma 8. By Definition 2, since only higher-priority tasks cause interference and because a response-time bound is known for each higher-priority task, we have  $x_k \leq W'_k(R_i^{apa})$ . The remainder of the proof proceeds analogously to the proof of Lemma 8 by replacing  $I_i^k(t) = \min(W_k(t), t - e_i + 1)$  with the refined interference bound  $I_i^{k'}(t) = \min(W'_k(t), t - e_i + 1)$ . ■

Again, schedulability of the entire task set may be established by applying Lemma 9 to each task in  $\tau$  in order of decreasing priority. The task set is schedulable under FP APA scheduling if Equation 18 yields a valid bound  $R_i^{apa} \leq d_i$  for each task  $T_i \in \tau$ . Lemma 9 thus establishes that Bertogna and Cirinei's RTA for FP scheduling with slack updates [10] is compatible with the reduction approaches proposed in Lemmas 3 and 5 and Algorithm 1.

### B.4 EDF RTA with Slack Updates

In this appendix, we establish that Bertogna and Cirinei's RTA for EDF scheduling with slack updates [10] is compatible with the reduction-based approach. We begin with a review of the original analysis.

*Review:* Bertogna and Cirinei's response-time analysis for EDF with slack updates follows in large parts the same idea as their slack-aware RTA for FP discussed in Appendix B.3, namely by exploiting slack bounds to derive more accurate workload and interference bounds.

However, since all tasks can potentially interfere with each other under EDF, the approach is slightly more involved under EDF. More precisely, as slack bounds have to be computed using response-time bounds for mutually interfering tasks, a

circular dependency exists, which is resolved with an additional fixed-point search across the entire task set.

To this end, Bertogna and Cirinei's slack-aware RTA for EDF proceeds in rounds. In each round, each task's response-time bound is computed using slack estimates based on results from the previous round. In the following, we let  $R_1^G(j), \dots, R_n^G(j)$  denote the response-time bounds obtained in the  $j^{\text{th}}$  round, and let  $s_1^{(j)}, \dots, s_n^{(j)}$  denote the slack estimates (for each task, respectively) used in the  $j^{\text{th}}$  round, i.e., the slack bounds used to obtain  $R_1^G(j), \dots, R_n^G(j)$ .

Initially, in the first round,  $s_i^{(1)} = 0$  for each task  $T_i \in \tau$ . After each round, slack updates are determined by the following rule, for  $j \geq 1$  and each task  $T_i \in \tau$ :

$$s_i^{(j+1)} = \begin{cases} d_i - R_i^G(j) & \text{if } R_i^G(j) \leq d_i, \\ 0 & \text{if } R_i^G(j) \text{ is undefined.} \end{cases}$$

Next, we review how to compute each  $R_i^G(j)$ . To exploit prior slack updates, Bertogna and Cirinei introduced a refined interference bound (for  $i \neq k$ )

$$I_i^{k'}(t) = \min(W_k'(t), \mathfrak{J}_i^k, t - e_i + 1), \quad (19)$$

where  $W_k'(t)$  is defined as given in Definition 2 and computed using the current slack estimates  $s_1^{(j)}, \dots, s_n^{(j)}$ , and where  $\mathfrak{J}_i^k$  is defined as given in Definition 3 (and also computed using the current slack estimates). If  $i = k$ , then  $I_i^{k'}(t) = 0$ .

Based on  $I_i^{k'}(R_i^G(j))$ , and hence implicitly also based on  $s_1^{(j)}, \dots, s_n^{(j)}$ , an upper bound on the response time of task  $T_i \in \tau$  is given by the least positive solution of the equation

$$R_i^G(j) = e_i + \frac{1}{m} \cdot \sum_{T_k \in \tau} I_i^{k'}(R_i^G(j)) \quad (20)$$

that is not greater than deadline  $d_i$ . If no such least fixed point  $R_i^G(j) \leq d_i$  exists, then  $R_i^G(j)$  is undefined (and  $s_i^{(j+1)} = 0$ ).

Bertogna and Cirinei's slack-aware RTA for EDF terminates after round  $j$  if  $s_i^{(j)} = s_i^{(j-1)}$  for each  $T_i \in \tau$ , i.e., when a global fixed point is reached. After termination in round  $j$ , the task set is deemed schedulable iff, for each  $T_i \in \tau$ ,  $R_i^G(j)$  is defined and  $R_i^G(j) \leq d_i$ , i.e., if the per-task fixed-point search did not fail to converge for any task.

*Proof of compatibility:* We establish that, if Equation 20, instantiated for each task's global-like subproblem, has a valid solution for each task, then all tasks are schedulable under EDF APA scheduling.

**Lemma 10:** *Let  $\tau$  denote a task set to be scheduled under EDF APA scheduling on a processor platform  $\pi$ . If there exist processor subsets  $\alpha'_1, \dots, \alpha'_n$  and least positive solutions  $R_1^{apa}, \dots, R_n^{apa}$  such that, for each  $T_i \in \tau$ ,  $\alpha'_i \subseteq \alpha_i$ ,  $R_i^{apa} \leq d_i$ , and  $R_i^{apa}$  is the least positive solution of Equation 21 below*

$$R_i^{apa} = e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} I_i^{k'}(R_i^{apa}), \quad (21)$$

where  $I_i^{k'}(R_i^{apa})$  is computed as defined in Equation 19 based on the response-times of all other tasks, then all tasks in  $\tau$  are

*schedulable and, for each task  $T_i \in \tau$ ,  $R_i^{apa}$  upper-bounds  $T_i$ 's response time in any schedule of  $\tau$  under EDF APA scheduling.*

*Remark:* A consistent solution vector  $R_1^{apa}, \dots, R_n^{apa}$  may be easily found using a global fixed-point search as outlined above in the review of the original global schedulability test; however, for the purpose of establishing correctness under APA scheduling, it is irrelevant how the solution  $R_1^{apa}, \dots, R_n^{apa}$  and the corresponding consistent slack values are determined.

*Proof:* The proof resembles the proof of Lemma 8. For the sake of clarity, we repeat the argument in full detail.

By contradiction. Suppose that there exist processor subsets  $\alpha'_1, \dots, \alpha'_n$  and a vector of least positive solutions  $R_1^{apa}, \dots, R_n^{apa}$  such that, for each  $T_i \in \tau$ ,  $\alpha'_i \subseteq \alpha_i$ ,  $R_i^{apa} \leq d_i$ , and  $R_i^{apa}$  is the least positive solution of Equation 21, but there exists an EDF schedule  $S$  of  $\tau$  under APA scheduling in which a job of some task  $T_i \in \tau$  has a response time exceeding  $R_i^{apa}$ , i.e., in which a job of  $T_i$  remains pending for more than  $R_i^{apa}$  time units.

Let  $t'$  denote the first point in time that a job of some task exceeds its task's claimed response-time bound in  $S$ . Let  $J$  denote (one of) the job(s) that exceed(s) the claimed response-time bound(s) at time  $t'$ , and let  $T_i \in \tau$  denote  $J$ 's task. Let  $t_r$  denote  $J$ 's release time. At time  $t'$ ,  $J$  has been pending (for the first time) for more than  $R_i^{apa}$  time units, i.e.,  $t' = t_r + R_i^{apa}$ .

Let  $X \subseteq [t_r, t_r + R_i^{apa})$  denote all times at which  $J$  is interfered with before time  $t_r + R_i^{apa}$ , i.e., at which  $J$  is not scheduled during  $[t_r, t_r + R_i^{apa})$ . Since  $J$  does not complete by time  $t_r + R_i^{apa}$ , it is scheduled for strictly less than  $e_i$  time units during  $[t_r, t_r + R_i^{apa})$ , and therefore  $|X| \geq t_r + R_i^{apa} - t_r - (e_i - 1) = R_i^{apa} - e_i + 1$ .

Let  $x_k$  denote the interference incurred by  $J$  due to task  $T_k$  before time  $t_r + R_i^{apa}$ , i.e., the total time that jobs of  $T_k$  execute on processors in  $\alpha_i$  during  $[t_r, t_r + R_i^{apa})$  while  $J$  is not scheduled. Because  $J$  is (among) the first job(s) to exceed the claimed response-time bound(s), no job of task  $T_k$  with a deadline before time  $t_r + R_i^{apa}$  is pending for longer than  $R_k^{apa}$  time units. Thus, based on Definition 2, the workload of  $T_k$  during  $[t_r, t_r + R_i^{apa})$  is bounded by  $W_k'(R_i^{apa})$ . We thus have

$$x_k \leq W_k'(R_i^{apa}), \quad (22)$$

for each  $T_k \in \tau$  since a task must be scheduled to interfere.

Furthermore, recall from the discussion in Appendix B.1 that  $\mathfrak{J}_i^k$  is a valid interference bound under both global and APA scheduling since the bound depends only on fixed tasks parameters, the assumption that interfering jobs of a task  $T_k$  comply with a known response-time bound, and the fact that jobs are prioritized according to EDF. Therefore, by Definition 3, we also have, for each  $T_k \in \tau$ ,

$$x_k \leq \mathfrak{J}_i^k. \quad (23)$$

Since all processors in  $\alpha_i$  are busy executing higher-priority jobs when  $J$  is not scheduled, we have

$$\sum_{T_k \in \tau} x_k = |X| \cdot |\alpha_i| \geq |\alpha_i| \cdot (R_i^{apa} - e_i + 1),$$

and therefore, by Lemmas 6 and 7, also

$$\sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, R_i^{apa} - e_i + 1) \geq |\alpha'_i| \cdot (R_i^{apa} - e_i + 1),$$

or, rearranged and strengthened to a strict inequality,

$$e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, R_i^{apa} - e_i + 1) > R_i^{apa}.$$

Additionally, by Inequality 23,  $x_k \leq \mathfrak{J}_i^k$  for any  $T_k \in \text{tasks}(\alpha'_i)$ , and thus also

$$e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, \mathfrak{J}_i^k, R_i^{apa} - e_i + 1) > R_i^{apa}.$$

It follows, by the definition of  $R_i^{apa}$  (Equation 21), that

$$\begin{aligned} e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, \mathfrak{J}_i^k, R_i^{apa} - e_i + 1) \\ > e_i + \frac{1}{|\alpha'_i|} \cdot \sum_{T_k \in \text{tasks}(\alpha'_i)} I_i^{k'}(R_i^{apa}), \end{aligned}$$

or, simplified,

$$\sum_{T_k \in \text{tasks}(\alpha'_i)} \min(x_k, \mathfrak{J}_i^k, R_i^{apa} - e_i + 1) > \sum_{T_k \in \text{tasks}(\alpha'_i)} I_i^{k'}(R_i^{apa}).$$

This implies that there exists a task  $T_k \in \text{tasks}(\alpha'_i)$  such that

$$\min(x_k, \mathfrak{J}_i^k, R_i^{apa} - e_i + 1) > I_i^{k'}(R_i^{apa}).$$

Since  $T_i$  does not interfere with itself (recall that  $J$  is the first job with a response time greater than  $R_i^{apa} \leq d_i$  and the restriction to constrained deadlines), it follows that  $x_k > 0$  is possible only if  $k \neq i$ . Therefore, by Equation 19, we have  $I_i^{k'}(t) = \min(W'_k(t), \mathfrak{J}_i^k, t - e_i + 1)$ , and thus

$$\begin{aligned} \min(x_k, \mathfrak{J}_i^k, R_i^{apa} - e_i + 1) \\ > \min(W'_k(R_i^{apa}), \mathfrak{J}_i^k, R_i^{apa} - e_i + 1). \end{aligned}$$

However, this implies  $x_k > W'_k(R_i^{apa})$ , which by Inequality 22 is impossible. Contradiction.

Therefore, all jobs of all tasks complete by their corresponding claimed response-time bounds and all deadlines are met.  $\blacksquare$

Lemma 10 establishes that Bertogna and Cirinei's RTA for EDF scheduling with slack updates [10] is compatible with the proposed reduction-based approach, in the sense that the global schedulability test can be reused for EDF APA scheduling by instantiating it for each task's global-like subproblem with a straightforward syntactic adjustment.