Multiprocessor Real-Time Scheduling with Arbitrary Processor Affinities: From Practice to Theory

Arpan Gujarati · **Felipe Cerqueira** · **Björn B. Brandenburg**

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 applications, 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 systems.

This paper presents the first analysis of multiprocessor scheduling with arbitrary processor affinities from a real-time perspective. 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 combined. Concerning the more general case of job-level dynamic priorities, it is shown that global and clustered scheduling are equivalent to multiprocessor real-time scheduling with arbitrary processor affinities.

The Linux push and pull scheduler is studied as a reference implementation and two approaches for the schedulability analysis of hard real-time tasks with arbitrary processor affinity masks are presented. In the first approach, the scheduling problem is reduced to "global-like" sub-problems to which existing global schedulability tests can be applied. The second approach is specifically based on response-time analysis and models the response-time computation as a linear optimization problem. The latter linear-programming-based approach has better runtime complexity than the former reduction-based approach. Schedulability experiments show the proposed techniques to be effective.

1 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

---

This paper is an extended version of a prior ECRTS 2013 paper. The extensions and new contributions are summarized in Section 1.1.

A. Gujarati · F. Cerqueira · B.B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)
Paul-Ehrlich-Straße G26, D-67663 Kaiserslautern, Germany.
E-mail: {arpanbg, felipec, bbb}@mpi-sws.org

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 in the literature (reviewed in Section 1.2) 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 a "global" scheduling policy is applied within each cluster.

Interestingly, many contemporary multiprocessor real-time operating systems, such as VxWorks, LynxOS, QNX, and 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* a process to an arbitrary subset of processors in the system, in the sense that a process can only migrate to (or be scheduled on) the processors that it is bound to.

Processor affinities are commonly used to boost application performance in throughput-oriented computing and to completely isolate real-time applications from non-real-time applications by assigning them to different cores (see Markatos and LeBlanc, 1992; Salehi et al, 1995; Alfieri, 1998; Foong et al, 2004, 2005; Jang and Jin, 2009). With processor affinities, it is also possible to address specific requirements of individual tasks. For example, cache-sensitive tasks with tight deadlines can be restricted to single processors to avoid migration overheads. In addition, in heterogeneous platforms with multiple types of cores, tasks can be assigned to specialized cores to reduce their power consumption and increase their performance (Reddy et al, 2011).

Processor affinities can also be used to realize global, partitioned, and clustered scheduling. For example, 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 the processor affinity 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 considered in the literature to date.

1.1 Contributions

In this paper, we present the first formal study of the scheduling problem with APAs (*APA scheduling* hereafter) in the context of the sporadic task model. In particular, this paper makes the following contributions.

- We show that APA scheduling is strictly more general than global and clustered scheduling combined with respect to the class of *job-level fixed-priority* policies (Section 3.1). This is an interesting theoretical property and provides a strong motivation for the use of APAs in multiprocessor real-time systems.

- We also investigate the generality of APA scheduling with respect to *job-level dynamic priority* policies and observe an equivalence of global and APA scheduling (Section 3.2).
- We propose the first schedulability analysis for sporadic tasksets with processor affinity restrictions (Section 4). Our *reduction-based* technique is generic in the sense that it can reuse any existing global schedulability analysis. To this end, we show how an APA scheduling problem can be reduced to "global-like" subproblems, which can then be analyzed using existing global schedulability analyses (Section 4.1).
- We argue that the accuracy of the reduction-based technique can be improved with an *exhaustive* search of the subproblem space (Section 4.2), albeit at the cost of an exponential number of invocations of the underlying global schedulability test. To overcome this cost, we propose a simple but effective search heuristic (Section 4.3).
- Focusing on the specific case of response-time analysis for fixed-priority schedulers, we derive a novel response-time analysis for APA scheduling based on *linear programming* (Section 5). We formally show that the proposed linear-programming-based analysis is at least as accurate as the exhaustive approach (Theorem 7), while reducing the time complexity of each iteration of the response-time analysis from exponential time to polynomial time.
- To evaluate if APA scheduling also offers improved schedulability in practice and to empirically explore how different schedulability analyses proposed in this paper relate to each other, we performed two sets of schedulability experiments, which we report on in Section 6. Notably, a comparison of the proposed APA schedulability analyses with a (simulation-based) APA "un-schedulability" test and a feasibility test showed that, in certain cases, the proposed analysis methods are already close to the feasibility limit (Section 6.3).

We believe this work is a significant first step towards the use of APAs in *predictable* hard real-time systems (*i.e.*, systems in which the timing correctness must be established *a priori*). Furthermore, we seek to establish a thorough foundation for future work on the APA scheduling problem, as we believe that scheduling with APAs merits increased attention from the real-time community. We continue with a brief discussion of related work in the field of multiprocessor real-time scheduling, the system model and notation used in the rest of the paper, and then continue with our analysis of APA scheduling.

Remark: This paper is an extended version of our previous paper, "Schedulability Analysis of the Linux Push and Pull Scheduler with Arbitrary Processor Affinities", which was published at the 25th Euromicro Conference on Real-Time Systems (Gujarati et al, 2013). Besides improving certain sections of the conference version for the sake of clarity, the following new contributions are made in this paper: **(i)** we establish an equivalence relation between APA scheduling and global and clustered scheduling with job-level dynamic priority policies (Section 3.2); **(ii)** we derive new response-time analysis and a faster linear-programming-based response-time analysis for APA scheduling (Section 5); **(iii)** we extended the empirical evaluation in Section 6 to report results of new schedulability experiments that evaluate the linear-programming-based analysis; and finally **(iv)** we incorporated feasibility and (simulation-based) "un-schedulability" tests for APA scheduling to provide better context for the observed schedulability results (Section 6.3).

1.2 Prior Work and Related Scheduling Problems

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

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 allows unrestricted migration of tasks across all processors (if required) while partitioned scheduling allows no migration at all (Davis and Burns, 2011b). Some notable hybrid scheduling algorithms that have been proposed include the aforementioned clustered scheduling (Baker and Baruah, 2007; Calandrino et al, 2007), *semi-partitioned* scheduling (*e.g.* see Anderson et al, 2005; Kato et al, 2009; Bado et al, 2012; Burns et al, 2012) and *restricted-migration* scheduling (*e.g.*, see Anderson et al, 2005; Dorin et al, 2010). We explain below how the aforementioned scheduling approaches differ from APA scheduling. In addition, we also compare APA scheduling to other scheduling problems of a similar structure.

Global, clustered, and partitioned scheduling APA scheduling *generalizes* global, clustered, and partitioned 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 an appropriate processor affinity assignment, a taskset can be modeled as a global, clustered, or partitioned taskset. (Section 3 formally proves the generality of APA scheduling.)

Semi-partitioned scheduling 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.

Restricted-migration scheduling APA scheduling, which restricts migrations to occur a-mong 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 can be combined with restricted-migration scheduling (Anderson et al, 2005; Dorin et al, 2010), and similar approaches could also be explored in the case of APA scheduling.

Virtual cluster-based scheduling While the general class of hierarchical scheduling algorithms is beyond the scope of this paper, we note that Easwaran et al (2009)'s work on hierarchical scheduling closely resembles APA scheduling. In particular, as under APA scheduling, the virtual cluster-based hierarchical scheduling scheme proposed by Easwaran et al (2009) also allows tasks to be assigned to overlapping physical clusters. However, APA scheduling is fundamentally different because it considers processor affinities as first-class entities. As a consequence, the APA scheduler and the schedulability analysis for APA scheduling expects the processor affinity assignment to be explicitly specified as part of the input workload, which allows the enforcement of arbitrary (*i.e.*, scheduling-unrelated) task placement restrictions.

Scheduling on unrelated heterogenous multiprocessors APA scheduling could also be understood as global scheduling on a (degenerate) *unrelated heterogeneous multiprocessor* (*e.g.*, see Funk, 2004), 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. While such platforms have primarily been studied in the context of partitioned scheduling to date (*e.g.*, Funk, 2004; Baruah, 2004; Andersson et al, 2010), Baruah and Brandenburg (2013) recently used this connection to derive feasibility tests for APA scheduling with implicit deadlines.

Non-real-time scheduling 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* (Gálvez et al, 2010; Leung and Li, 2008), *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).

Orthogonal to the degree of migration allowed, scheduling algorithms also have a choice of how to prioritize different jobs or tasks in a taskset 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.

An FP policy assigns a unique priority to each task; *e.g.*, the classic *Rate Monotonic* (RM) (Liu and Layland, 1973) and *Deadline Monotonic* (DM) (Leung and Whitehead, 1982; Audsley et al, 1991) 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* (EDF) policy (Liu and Layland, 1973). A JLDP policy allows a job to have distinct priorities during its lifetime; a prominent example in this category is the *Least Laxity First* policy (Dertouzos and Mok, 1989). APA scheduling can be combined with any of these priority assignment policies.

In this paper, we compare APA scheduling with global, clustered and partitioned scheduling with both JLFP and JLDP policies. However, we restrict our focus to JLFP and FP policies in our schedulability analysis framework, since such policies can be implemented with low overheads and are more frequently used in practice. For instance, most proprietary real-time operating systems use fixed-priority schedulers and Linux has recently added support for a JLFP policy, *i.e.*, EDF using the SCHED_DEADLINE class (Lelli et al, 2011). In that regard, we propose generic schedulability analysis techniques for APA scheduling that apply to both FP and JLFP scheduling (see Section 4), and also propose a concrete response-time analysis that is specific to FP scheduling (see Section 5). Next, we briefly formalize our system model before providing a formal definition of APA scheduling.

1.3 System Model

We consider the problem of scheduling a set of n real-time tasks $\tau = \{T_1, \ldots, T_n\}$ on a set of m identical processors $\pi = \{\Pi_1, \Pi_2, \ldots, \Pi_m\}$. We adopt the classic *sporadic task model* (Mok, 1983), 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 δ_i of a task T_i is $e_i/\min(d_i, p_i)$.

Each task T_i also has an associated processor affinity α_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 α_i is static, *i.e.*, processor affinities do not change over time. We define the joint processor affinity $cpus(\gamma)$ of a taskset γ as the set of processors on which at least one task in γ can be scheduled. Similarly, for a set of processors ρ, $tasks(\rho)$ defines the set of tasks that can be scheduled on at least one processor in ρ.

$$cpus(\gamma) = \bigcup_{\forall T_i \in \gamma} \alpha_i \tag{1}$$

$$tasks(\rho) = \{T_i \mid \alpha_i \cap \rho \neq \emptyset\} \tag{2}$$

A task T_k can (directly) *interfere* with another task T_i, *i.e.*, delay T_i's execution, only if α_k overlaps with α_i. We let I_i denote the set of all such tasks in τ whose processor affinities overlap with α_i. In general, the exact interfering taskset depends on the scheduling policy. Therefore, we define I_i^A as the interfering taskset 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 $prio_k$ denote T_k's fixed priority, where $prio_k > prio_i$ implies that T_k has a higher priority than T_i (*i.e.*, T_k can preempt T_i), then

$$I_i^{FP} = \{T_k \mid prio_k > prio_i \wedge \alpha_k \cap \alpha_i \neq \emptyset\}. \tag{3}$$

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 taskset*. A task $T_i \in \tau$ is schedulable on the processor platform π if it can be shown that no job of T_i ever misses its deadline. A taskset τ is schedulable on the processor platform π if all tasks in τ are schedulable on π.

In addition, when comparing scheduling algorithms (in Section 3), we use the concepts of *dominance* and *equivalence*. For any two scheduling algorithms $\mathscr{A}$ and $\mathscr{B}$, $\mathscr{A}$ is *equivalent* to $\mathscr{B}$ if for any real-time taskset τ (as defined by the aforementioned sporadic task model), τ is schedulable under $\mathscr{A}$ *iff* τ is schedulable under $\mathscr{B}$. In contrast, $\mathscr{A}$ *dominates* $\mathscr{B}$ if for any taskset τ schedulable under $\mathscr{B}$, τ is also schedulable under $\mathscr{A}$. Further, $\mathscr{A}$ *strictly dominates* $\mathscr{B}$ if $\mathscr{A}$ dominates $\mathscr{B}$ *and* there exists at least one taskset τ' such that τ' is schedulable under $\mathscr{A}$ but not under $\mathscr{B}$. We analogously apply the concepts of strict dominance and equivalence to classes of scheduling algorithms (such as FP and JLFP schedulers).

1.4 Paper Organization

The rest of this paper is structured as follows. In Section 2 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 Section 3, we compare APA scheduling with global, partitioned, and clustered scheduling from a schedulability perspective for both JLFP and JLDP policies. In Section 4, we present generic schedulability analysis for APA scheduling. In Section 5, we present response-time analysis for APA scheduling with fixed priorities that uses a novel linear programming technique to improve the runtime complexity of the analysis algorithm. We discuss the evaluation results of schedulability experiments in Section 6. Lastly, Section 7 gives concluding remarks.

2 Push and Pull Scheduling in Linux

The Linux kernel uses 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. In the remainder of this section, we review the Linux scheduler implementation of global and APA scheduling to illustrate the similarities between these two approaches. Based on these similarities, we later derive schedulability analysis techniques for APA scheduling in Section 4.

2.1 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 $prio_i(t)$ 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:*

$$\forall T_s \in S(t),\ prio_b(t) \leq prio_s(t) \wedge |S(t)| = m. \tag{4}$$

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 Π_s denote the source and let Π_t denote the target processor, and let T_m be the task to be migrated. A *push* operation is performed by Π_s on T_m if T_m becomes available for execution on Π_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 Π_t) such that the task currently assigned to Π_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 Π_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 Π_s for a task T_m assigned to Π_s's runqueue such that T_m is backlogged and T_m's priority exceeds that of all local tasks in Π_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 Π_s is currently serving a low-priority task T_l when a higher-priority task T_h becomes available for execution on Π_s (*i.e.*, processor Π_s handles the interrupt that causes T_h to release a job). Then Π_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 Π_t exists for T_l at the time of preemption, T_l will remain queued on Π_s until it is discovered later by a pull operation (or until T_h's job completes and Π_s becomes available again).

It is important to note that 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 Π_t, it can only be "dislodged" by the arrival of a higher-priority task on Π_t, either due to a push operation targeting Π_t or due to an interrupt handled by Π_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). Linux makes no assumption on which processor handles interrupts—that is, a job may be released on potentially any processor (ignoring affinity restrictions). The scheduler is then responsible for assigning an arriving task to the appropriate processor.

2.2 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 α_b. Let $S'(t)$ be the set of tasks that are scheduled on any processors in α_b at time t. If $prio_i(t)$ denotes the priority of a task T_i at time t, then under APA scheduling:*

$$\forall T_s \in S'(t),\ prio_b(t) \leq prio_s(t) \wedge |S'(t)| = |\alpha_b|. \tag{5}$$

A key feature of Linux's scheduler is that push and pull operations seamlessly generalize to APA scheduling. A push operation on Π_s migrates T_m from Π_s to Π_t only if $\Pi_t \in \alpha_m$. Similarly, a pull operation on Π_t pulls T_m from Π_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 α_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 α_m are occupied by higher-priority tasks, even if some task $T_h \in S'(t)$ could be scheduled on another processor Π_x not part of α_m (*i.e.*, in the worst case, if $\Pi_x \in \alpha_h$ and $\Pi_x \notin \alpha_m$, then Π_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 is not ideal from a schedulability point of view, it has the advantage of simplifying the implementation.

From the definitions of the global and APA scheduling invariants, we can easily see 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 a 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 that APA JLFP scheduling strictly dominates global, clustered, and partitioned JLFP scheduling in Section 3 below, and then in Sections 4 and 5 present schedulability tests applicable to all schedulers that guarantee the APA scheduling invariant given in Equation 5.

3 Generality of APA Scheduling

Recall from Section 1 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 also useful from a timeliness point of view.

3.1 APA Scheduling with JLFP Policies

As discussed in Sections 1 and 2.2, 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). APA scheduling can thus emulate global, clustered, and partitioned scheduling by assigning every task in the taskset an appropriate processor affinity, which we summarize with the following lemma.

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

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 α_i and α_k such that $\alpha_i \cap \alpha_k \neq \emptyset$. As a result, as we show next, there exist tasksets that are schedulable under APA scheduling, but infeasible under global, clustered, and partitioned scheduling.

Consider the taskset described in Table 1, 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. In the following, we try to schedule this taskset using global, partitioned, and APA JLFP scheduling. We do not explicitly consider

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 1 Workload parameters used in Theorem 1.

Task	α_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	$\{\Pi_1, \Pi_2\}$

Table 2 Processor affinity assignment.

clustered scheduling because, for two processors, clustered scheduling reduces to either global or partitioned scheduling. We begin with global scheduling.

Lemma 2 *The taskset given in Table 1 is infeasible on a two-processor system under global JLFP scheduling with any JLFP rule.*

Proof Refer to Figure 1 for an illustration of the following discussion. 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 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. □

Next, we establish that the taskset cannot be partitioned.

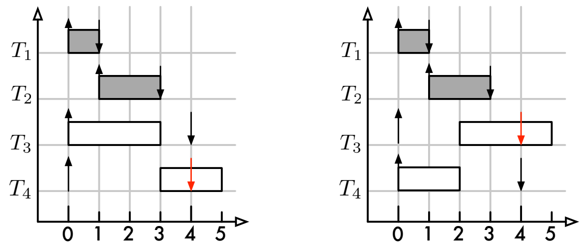
Lemma 3 *The taskset given in Table 1 cannot be partitioned onto a two-processor system.*

Proof 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. □

Finally, we observe that the taskset *is* schedulable if suitable per-task processor affinities can be assigned.

Lemma 4 *The taskset given in Table 1 is feasible on a two-processor system under APA JLFP scheduling.*

Proof 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 taskset. Therefore, using processor affinities as given in Table 2, we partition tasks T_1, T_2, T_3, T_4, T_5, and T_6, but allow task T_7 to migrate. The taskset 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 $\Gamma = [t_a, t_d)$



(a) T_4's deadline miss. (b) T_3's deadline miss.

Fig. 1 Global JLFP schedules of tasks T_1, T_2, T_3, and T_4 in Table 1. The small up-arrows and down-arrows indicate job-arrivals and deadlines respectively. Jobs executing on processor Π_1 are in shaded in grey and jobs executing on processor Π_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.

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 Π_2 and Π_1 in sequence. First, we bound the minimum time for which J_7 can execute on Π_2. Then, to ensure T_7's schedulability, we argue that J_7 can always satisfy its remaining processor demand on Π_1.

We use techniques introduced by Baruah (2007) 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 Γ, the maximum interference incurred by J_7 on Π_2 due to tasks T_2, T_4, and T_6 is bounded by $2+2+5001 = 5005$. (The exact interference $\mathscr{I}$ varies with the inter-arrival times of jobs.) If $\mathscr{I} \leq 5000$, then J_7 can be scheduled successfully on Π_2 itself. However, if $\mathscr{I} > 5000$, J_7 must satisfy its remaining demand on Π_1; *i.e.*, if $\mathscr{I} = 5000+\delta$ where $\delta \in [1,5]$, then J_7 must execute on processor Π_1 for at least δ time units.

Let Γ' denote the cumulative interval(s) in Γ when jobs of T_6 interfere with J_7 on Π_2. Since the contribution of tasks T_2 and T_4 to $\mathscr{I}$ is at most $2+2=4$, the contribution of T_6 to $\mathscr{I}$ is at least $4996+\delta$ (recall that $\mathscr{I} = 5000+\delta$). 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 align with t_a but precedes t_a). In either case, Γ' consists of at least one contiguous interval $\Gamma'' \in \Gamma'$ of length $2498+\delta/2$. However, in any contiguous interval of length $2498+\delta/2$, Π_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 Π_2 is continuously unavailable during Γ'', Π_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 Π_1. Therefore, T_7 is schedulable and the taskset is schedulable under APA JLFP scheduling. □

Taken together, Lemmas 1–4 show that a careful choice of processor affinities can render a taskset feasible when global, partitioned, and clustered JLFP scheduling fails. We summarize this observation with the following theorem.

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

Proof By Lemma 1, APA JLFP scheduling is at least as powerful as global, partitioned, and clustered JLFP scheduling combined. By Lemmas 2–4, there exists a taskset that can be scheduled under APA JFLP scheduling, but not under global, clustered, or partitioned JFLP scheduling. The claimed strict dominance follows. □

Theorem 1 provides further motivation to explore the benefits of APA scheduling in a real-time context. Next, we discuss the more general case of JLDP policies.

3.2 APA Scheduling with JLDP Policies

Two important results regarding global JLDP scheduling are as follows: **(i)** there exist global JLDP policies that are optimal for implicit-deadline tasks (Baruah et al, 1996); and **(ii)** optimal online scheduling of constrained-deadline tasks (and therefore, also of tasks with arbitrary deadlines) is generally impossible (Fisher et al, 2010). Thus, while the existence of optimal APA JLDP policies for implicit-deadline tasks trivially follows from Lemma 1, we are more interested in understanding how APA scheduling with JLDP policies fares for constrained-deadline tasks. In this respect, we state the following theorem.

Theorem 2 *Global JLDP scheduling is equivalent to APA JLDP scheduling for tasks with constrained deadlines.*

Proof From Lemma 1, any taskset that is schedulable under global scheduling is also schedulable under APA scheduling. Thus, APA JLDP scheduling is at least as general as global JLDP scheduling.

Next, we show that given any APA JLDP scheduler $\mathscr{A}$ and a real-time workload that is schedulable using $\mathscr{A}$, a global JLDP scheduler $\mathscr{G}$ can always be constructed that successfully schedules the same real-time workload as well. In particular, $\mathscr{G}$ simulates $\mathscr{A}$ throughout the execution of the workload and uses the results of this simulation to make global scheduling decisions. In the following, let τ be the real-time workload under consideration, which is to be scheduled on a multiprocessor platform π.

Since $\mathscr{G}$ precisely knows the set of tasks scheduled as per $\mathscr{A}$ at any time t, it uses this information to assign priorities to the ready tasks. Assume there are only two distinct priority levels, HI and LO, such that a task with priority HI is considered to have a higher-priority than a task with priority LO. Then, $\mathscr{G}$ assigns priority HI to all tasks that are ready to execute and that are also scheduled on some processor as per $\mathscr{A}$ at time t. The remaining ready tasks are assigned the priority LO. The global JLDP scheduler $\mathscr{G}$ then schedules the $|\pi|$ highest priority ready tasks (with ties in priority broken arbitrarily).

The above priority assignment rule and the policy to (at any time t) schedule the $|\pi|$ highest-priority ready tasks guarantees that every job scheduled under $\mathscr{A}$ at time t is also scheduled under $\mathscr{G}$ at time t (unless it has already finished its execution). Therefore, if a workload is schedulable under $\mathscr{A}$, then it is also schedulable under $\mathscr{G}$. Thus, global JLDP scheduling is as general as APA JLDP scheduling. □

While it may admittedly be impractical for a global JLDP scheduler to simulate an APA JLDP scheduler at runtime (due to performance reasons), this technique suffices to establish the equivalence of the two classes of scheduling algorithms. In comparison with partitioned scheduling, APA scheduling is of course more general irrespective of the employed priority assignment policy (because of the existence of tasksets that cannot be partitioned).

In the following, since FP and JLFP policies are used more frequently than JLDP policies in practice, and since global JLDP scheduling and APA JLDP are (at least theoretically) equivalent, we emphasize Theorem 1 and therefore restrict our focus to FP and JLFP policies when deriving schedulability analyses for APA scheduling in Sections 4 and 5.

4 Schedulability Analysis

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 tasksets with APAs has been proposed to date. In this section, we apply the ideas from Section 2 that relate APA scheduling to the well-studied global scheduling problem, and propose simple and efficient techniques to analyze tasksets for APA 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. The section is divided into three parts. We start with a simple method for analyzing tasksets 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.

4.1 Reduction to Subproblems

Recall from Sections 2 and 3 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 α_i, and a reduced taskset $tasks(\alpha_i)$, APA scheduling reduces to global scheduling. 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 α_5, then it is also schedulable using APA scheduling on the processor platform π. 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 ρ.

Lemma 5 *If a task $T_i \in tasks(\alpha_i)$ is schedulable when the reduced taskset $tasks(\alpha_i)$ is globally scheduled on the reduced processor platform α_i using a JLFP policy A, then T_i is also schedulable under APA scheduling of τ on the processor platform π using the same JLFP policy A.*

Proof By contradiction. Suppose a task $T_i \in tasks(\alpha_i)$ is schedulable under global scheduling on the processor platform α_i using a JLFP policy A, but it is not schedulable under APA scheduling on the processor platform π 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.*, α_i) are busy executing 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 α_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 $e_i + |\Theta_i(t_a, t_a + d_i)| > d_i$.

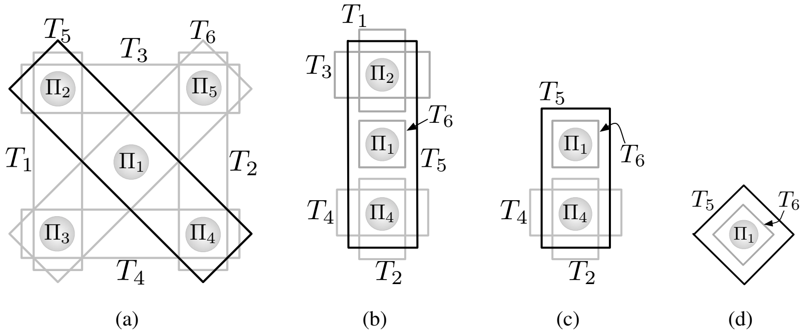


Fig. 2 Four scheduling problems (a), (b), (c), and (d) are illustrated here. The circles represent the processors and the rectangles represent the tasks and their associated processor affinities, *e.g.*, problem (a) consists of the processor set $\pi = \{\Pi_1, \Pi_2, \Pi_3, \Pi_4, \Pi_5\}$ and the taskset $\tau = \{T_1, T_2, T_3, T_4, T_5, T_6\}$. 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.

Since T_i is not schedulable under APA scheduling on the processor platform π, 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.*,

$$\exists t_a : e_i + |\Theta_i^{APA}(t_a, t_d)| > d_i. \tag{6}$$

However, since $T_i \in \tau$ is schedulable under global scheduling on the reduced processor platform α_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.*,

$$\forall t_a : e_i + |\Theta_i^G(t_a, t_d)| \leq d_i. \tag{7}$$

The work that comprises $\Theta_i^{APA}(t_a, t_d)$ is computed upon α_i. $\Theta_i^G(t_a, t_d)$ is computed upon all processors in the processor platform, which is equal to α_i in this case. Also, by construction, under both APA and global scheduling, the same set of tasks keeps processors in α_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 6 and 7 cannot both be true simultaneously. □

Using the equivalence from Lemma 5, 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. For this purpose, we assume the availability of a generic test *GlobalAnalysis*(A, T_i, π, ζ_i) to analyze the schedulability of a single task, where A is the scheduling policy, T_i is the task to be analyzed, π is the processor set on which the task is to be scheduled, and ζ_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 τ are schedulable; we are only concerned with the schedulability of task T_i. Using this interface we define a simple method to analyze tasksets for APA scheduling. The key idea is to identify for each task an "equivalent" global subproblem, and to then invoke *GlobalAnalysis*(A, T_i, π, ζ_i) on that subproblem.

Task	e_i	d_i	p_i	α_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	$\{\Pi_1, \Pi_3, \Pi_5\}$

Table 3 Workload parameters and processor affinities of the taskset discussed in Example 1.

Lemma 6 *A taskset τ is schedulable on a processor set π under APA scheduling using a JLFP policy A if*

$$\bigwedge_{\forall T_i \in \tau} GlobalAnalysis(A, T_i, \alpha_i, I_i^A). \tag{8}$$

Proof The analysis checks schedulability of each task $T_i \in \tau$ under global scheduling on processor platform α_i. From Lemma 5, if each task $T_i \in \tau$ is schedulable on the corresponding reduced processor platform α_i, then T_i is also schedulable on the processor platform π under APA scheduling. Therefore, the entire taskset τ is schedulable under APA scheduling with policy A on the processor platform π if for each task T_i the implied global-like subproblem witnesses the schedulability of T_i. □

The analysis technique in Lemma 6 is a straightforward way to reuse global schedulability analysis for analyzing tasksets with APAs, *i.e.*, tasksets 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 taskset (*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 α_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 that task T_i is not schedulable with the given processor affinity α_i, but claims that it is schedulable with a different processor affinity $\alpha_i' \subset \alpha_i$, *i.e.*, the result of the schedulability analysis in Lemma 6 may vary for the same task if reduced to different subproblems.

Example 1 *Consider the taskset described in Table 3, 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: $\forall i < k$, $prio_i > prio_k$. To analyze the taskset, we define $GlobalAnalysis(FP, T_i, \alpha_i, I_i^{FP})$ in Lemma 6 to denote a modified version of the response-time analysis for global fixed-priority scheduling (Bertogna and Cirinei, 2007), as later discussed in more detail in Section 6. Task T_5 fails the $GlobalAnalysis(FP, T_5, \alpha_5, I_5^{FP})$ test with the given processor affinity, i.e, $\alpha_5 = \{\Pi_1, \Pi_2, \Pi_4\}$ (see Figure 2(b) for the corresponding subproblem). However, if applied to a different* subset *of the processor affinity, i.e., $\alpha_5' = \{\Pi_1, \Pi_4\}$ as shown in Figure 2(c), T_5 is deemed schedulable by the test. Note that on the processor platform α_5, all four higher priority tasks can interfere with T_5, but on the processor platform α_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 α_5', but not on α_5.*

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

4.2 Exhaustive Reduction

Lemma 7 *If a task $T_i \in \tau$ is schedulable under APA scheduling with the processor affinity $\alpha_i' \subset \alpha_i$ and taskset τ, then T_i is also schedulable under APA scheduling with the affinity α_i and taskset τ.*

Proof By contradiction, analogous to Lemma 5. Recall from the proof of Lemma 5 that $\Theta_i(t_1,t_2)$ denotes the sub-interval of interference during which all processors in α_i are busy executing tasks other than T_i. Similarly, we define $\Theta_i'(t_1,t_2)$ over all processors in α_i'. We assume that T_i is not schedulable under APA scheduling with processor affinity α_i and taskset τ 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

$$\exists t_a : e_i + |\Theta_i(t_a, t_a + d_i)| > d_i, \tag{9}$$

$$\forall t_a : e_i + |\Theta_i'(t_a, t_a + d_i)| \leq d_i. \tag{10}$$

For any arbitrary, fixed arrival sequence, at any time instant, if all processors in α_i are busy executing tasks other than T_i, then all processors in α_i' must also be executing tasks other than T_i since $\alpha_i' \subseteq \alpha_i$. 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 9 and 10 cannot be true simultaneously. □

In Example 1, the schedulability test could not claim task T_5 to be schedulable with a processor affinity of α_5. However, the test claimed that the same task T_5, assuming a reduced processor affinity of $\alpha_5' \subset \alpha_5$, is schedulable. Note that this example does not contradict Lemma 7. While the result of Lemma 7 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 7 and the simple test in Lemma 6 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 taskset τ is deemed schedulable if all tasks $T_i \in \tau$ pass the test. Recall from Lemma 7 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 α_i. However, it does *not* require modifying the processor affinity of T_i from α_i to S in the actual system; rather, the reduction is merely an analysis assumption. In particular, while analyzing a task T_i, the processor affinities of all other tasks must remain unchanged.

Theorem 3 *A taskset τ is schedulable on a processor set π under APA scheduling using a JLFP policy A if*

$$\bigwedge_{T_i \in \tau} \Big(\bigvee_{\substack{S_i \subseteq \alpha_i \\ S_i \neq \emptyset}} GlobalAnalysis(A, T_i, S_i, I_i^A \cap tasks(S_i)) \Big). \tag{11}$$

Proof 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 by Lemma 5, T_i is also schedulable under APA scheduling with the processor affinity S_i and the policy A. From Lemma 7, 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 α_i. Therefore, if corresponding subsets exist for every task in τ, the taskset τ is schedulable on the processor set π 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 α_i. However, since $|\alpha_i|$ is bounded by m, and since the schedulability tests have to be run for all the tasks in the taskset, in the worst case, the algorithm requires $O(n \cdot 2^m)$ invocations of $GlobalAnalysis(A, T_i, \alpha_i, I_i^A)$. Despite the exponential complexity, we observed in our experiments that an exhaustive approach is still feasible for contemporary embedded multiprocessors with up to five processors. However, for multiprocessor systems with a higher number of processors, we need an alternative algorithm that does not analyze all possible subsets of a task's processor affinity. Instead, in the next section, we propose a heuristic to identify and test only a few "promising" subsets for each task.

4.3 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 taskset from Example 1 (Table 3). Since the schedulability of tasks* $T_1, T_2, \ldots, T_5$ *has already been established in Example 1, we carry out analysis for task* T_6 *in this example.* T_6 *fails* $GlobalAnalysis(FP, T_6, \alpha_6, I_6^{FP})$ *with the processor affinity as given in Figure 2(b), i.e,* $\alpha_6 = \{\Pi_1, \Pi_3, \Pi_5\}$. *Therefore, we seek an appropriate subset* $\alpha_6' \subset \alpha_6$ *such that* T_6 *is claimed to be schedulable on processor platform* α_6'. *However, unlike the algorithm given in Theorem 3, we select only promising subsets of* α_6. *To this end, in each iteration, we remove the processor that contributes the most to the total interference.*

Iteration 1 $\alpha_6 = \{\Pi_1, \Pi_3, \Pi_5\}$. *The removal candidates in* α_6 *are processors* Π_1, Π_3 *and* Π_5. *Removing processor* Π_1 *leads to removal of task* T_5, *removing processor* Π_3 *leads to removal of tasks* $\{T_1, T_4\}$ *and removing processor* Π_5 *leads to removal of tasks* $\{T_2, T_3\}$ *from* I_6^{FP}. *We choose to remove processor* Π_3 *because tasks* $\{T_1, T_4\}$ *contribute most to the total interference on task* T_6. *But* T_6 *still fails the schedulability test.*

Iteration 2 $\alpha_6' = \{\Pi_1, \Pi_5\}$. *The removal candidates in* α_6' *are processors* Π_1 *and* Π_5. *Removing processor* Π_1 *leads to removal of task* T_5 *and removing processor* Π_5 *leads to removal of tasks* $\{T_2, T_3\}$ *from* $I_6^{FP'}$. *We choose to remove processor* Π_5 *because tasks* $\{T_2, T_3\}$ *contribute more to the total interference on task* T_6 *than task* T_5. *The new subset is thus* $\alpha_6'' = \{\Pi_1\}$ *and task* T_6 *passes the schedulability test. Therefore,* T_6 *is schedulable under APA scheduling with an FP policy.*

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:         $t(c) \leftarrow tasks(\alpha_i^k) \setminus tasks(\alpha_i^k \setminus c)\}$
14:         $\Delta(c) \leftarrow \sum_{T_x \in t(c)} (\left\lceil \frac{d_i}{p_x} \right\rceil + 1) e_x$
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** $(\alpha_i^{k+1} = \alpha_i^k) \vee (\alpha_i^{k+1} = \phi)$

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

Algorithm 1 shows the pseudo-code for heuristically determining subsets of α_i and then invoking global analysis on those subsets. α_i^k is the new subset to be analyzed in the beginning of the k^{th} iteration and I_i^k is the corresponding interfering taskset. RC denotes the set of removal candidates. A removal candidate is a set of processors c such that, if c is removed from α_i^k to obtain the new subset α_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 α_i^k should lead 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 α_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).

$$\frac{\Delta(c)}{|c|} = \frac{1}{|c|} \cdot \left(\sum_{T_x \in t(c)} \left(\left\lceil \frac{d_i}{p_x} \right\rceil + 1 \right) \cdot e_x \right) \tag{12}$$

For a removal candidate c, the estimated demand reduction per processor quantifies the approximate reduction in total interference after the k^{th} iteration, if c was removed from α_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 (*i.e.*, 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 α_i^{k+1} w.r.t. α_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 taskset τ 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, because it is a heuristic-based algorithm and does not exhaustively evaluate all possible subsets, it may still miss out on prospective subsets that may yield positive results. We explore this tradeoff empirically in Section 6.

4.4 Further Restricting Affinities

Although the schedulability analysis that we discussed does not modify processor affinities, there are certain advantages in doing so. Given an initial affinity assignment, the operating system is conceptually free to further restrict the affinity of a task (although we are not aware of any system that does so), for example to avoid migrations (as in partitioned scheduling), or to better exploit the cache hierarchy (as in clustered scheduling).

Note that the proposed schedulability analysis can still be applied to reduced affinities. However, finding a good set of reduced affinities can be difficult as the initial affinity assignment may restrict the bin-packing solution space, limiting the efficacy of conventional task partitioning heuristics. We leave this problem of affinity-aware partitioning as future work.

Based on the principles developed in this section, we next present response-time analysis for APA scheduling with fixed-priorities and constrained deadlines. While the following techniques could analogously also be applied to obtain response-time analysis for APA scheduling with JLFP policies or arbitrary deadlines, we focus on FP policies with constrained deadlines for the sake of brevity.

5 Response-Time Analysis

Response-time analyses (RTA) for real-time workloads typically use fixed-point iteration methods to compute upper bounds on the response-times of all tasks (Joseph and Pandya, 1986; Audsley et al, 1993; Lundberg, 1998; Andersson and Jonsson, 2000; Harbour and Palencia, 2003; Palencia and Harbour, 2005; Bertogna and Cirinei, 2007; Guan et al, 2009). In this section, we first illustrate how to perform RTA for APA scheduling using the idea of reducing an APA scheduling problem to multiple "global-like" subproblems (as proposed in Section 4.2). Then, to improve the runtime complexity of the schedulability analysis, we present a novel approach based on linear programming (LP), in which a task's response-time is bounded by solving an LP in each iteration to determine the worst-possible interference (Section 5.3).

Both approaches discussed in this section extend the RTA for constrained-deadline tasks proposed by Bertogna and Cirinei (2007), which we review in Section 5.1. However, the proposed approach can also be applied to other multiprocessor RTAs due to the common structure of all response-time analyses.

5.1 Response-time Analysis by Bertogna and Cirinei (2007)

The maximum response time r_k of a task T_k is defined as the maximum time taken by any of task T_k's jobs to finish its execution. The analysis of Bertogna and Cirinei (2007) derives upper bounds on this response time, denote by r_k^{ub}, and is based on the concepts of *workload* and *interference*. The *workload* $W_k(t)$ of a task T_k is the maximum duration for which task T_k can execute in any interval of length t.

The workload is based on the number of jobs $n_k(t)$ that contribute with an entire WCET in any interval of length t, which is given by

$$n_k(t) = \left\lfloor \frac{t + d_k - e_k}{p_k} \right\rfloor. \tag{13}$$

Given $n_k(t)$, the workload of a task T_k is defined as follows:

$$W_k(t) = n_k(t) \cdot e_k + \min(e_k,\ t + d_k - e_k - n_k(t) \cdot p_k). \tag{14}$$

The *interference* $H_k^i(t)$ of a higher-priority task T_i on the analyzed lower-priority task T_k (in any interval of length t) is the cumulative length of all sub-intervals in which T_k is backlogged but cannot be scheduled on any processor while T_i is executing. The interference depends on the workload of the interfering task.

$$H_k^i(t) = \min(W_i(t),\ t - e_k + 1) \tag{15}$$

The complete RTA for global scheduling as derived by Bertogna *et al*. using the above definitions of workload and interference is stated in the following theorem. For this theorem and for the remainder of this paper, we let hp_k denote the set of tasks with priorities higher than or equal to T_k's priority, irrespective of whether the processor affinities of tasks in hp_k overlap (or not) with α_k.

Theorem 4 (Theorem 7 in (Bertogna and Cirinei, 2007)) *An upper bound on the response time of task T_k in a multiprocessor system scheduled under global scheduling with fixed priorities can be derived by the fixed-point iteration on the value r_k^{ub} of the following expression, starting with $r_k^{ub} = e_k$:*

$$r_k^{ub} \leftarrow e_k + \left\lfloor \frac{1}{m} \cdot \sum_{\forall T_i \in hp_k} H_k^i(r_k^{ub}) \right\rfloor. \tag{16}$$

Next, we discuss in detail two approaches to adopt this RTA for schedulability analysis of workloads under APA scheduling.

5.2 Response-Time Analysis for APA Scheduling by Reduction to Subproblems

The RTA for APA scheduling with fixed-priorities is a straightforward extension of the generic reduction-based schedulability analysis discussed in Section 4.2 and is stated in the following theorem.

Theorem 5 *An upper bound on the response time of task T_k in a multiprocessor system scheduled under APA scheduling with fixed priorities can be derived by the fixed point iteration on the value r_k^{ub} of the following expression, starting with $r_k^{ub} = e_k$:*

$$r_k^{ub} \leftarrow e_k + \min_{s \subseteq \alpha_k \wedge s \neq \emptyset} \left\lfloor \frac{1}{|s|} \cdot \sum_{T_i \in hp_k \wedge \alpha_i \cap s \neq \emptyset} H_k^i(r_k^{ub}) \right\rfloor . \quad (17)$$

Proof Analogous to Theorem 3. Under work-conserving preemptive scheduling, a task's response-time does not increase if either processors are added or interference from higher-priority tasks is reduced. Hence, T_k's response-time bound in any "global-like" subproblem bounds T_k's response time in the actual APA schedule. □

A single iteration based on Equation 17 has an exponential time complexity $O(2^{|\alpha_k|})$ because it requires checking every subset of α_k. In the following, we show that by modeling Equation 17 as a linear program (LP), polynomial time complexity can be achieved (w.r.t. one iteration of Equation 17). The improved runtime complexity is validated in the evaluation Section 6, where we can observe a noticeable difference between the scalability of the LP-based analysis and the analysis stated in Theorem 5.[1]

5.3 Response-time Analysis using Linear Programming

In this section, we propose an LP-based response-time analysis for APA scheduling with fixed priorities, which dominates the response-time analysis presented in the earlier Section 5.2. In particular, just like in Theorem 4, the upper bound r_k^{ub} on the response-time of task T_k is calculated through a fixed-point iteration, but with intermediate values obtained by solving an LP. As we show later, the advantage of modeling the problem as an LP is that when deriving a response-time bound, all subsets of task T_k's processor affinity need not be explicitly considered.

Recently, LP-based approaches have been adopted to address various problems related to real-time scheduling, including schedulability and feasibility analyses (*e.g.*, Lisper and Mellgren (2001); Baruah and Bini (2008); Zeng and Di Natale (2010)). However, to the best of our knowledge, LPs have not been previously used to derive bounds on interference under global or APA scheduling.

In the following section, before giving details about the LP-based analysis, we derive some properties that form the basis for our approach.

[1] The introduction of the LP, however, does not change the time complexity of the complete response-time analysis, which in corner cases still remains computationally intractable (see Eisenbrand and Rothvoß, 2008, 2010).

5.3.1 Bounding the Response-time under APA Scheduling

Before stating the LP formulation, let us present how to derive an upper-bound on the response time of a task assuming arbitrary processor affinities. Consider the execution of a task T_k in any valid schedule. In order to satisfy the temporal constraints, deadlines must be met in the presence of the maximum possible interference by higher-priority tasks.

To analyze the worst-case response time of task T_k, let us consider the interference incurred by some job of T_k due to higher-priority tasks in a time window of size t, starting upon the arrival of a job of task T_k. In the following, we show that the response time of a task is constrained by certain invariants that hold in every possible schedule of τ. Those invariants will be used later as constraints in the LP (see Section 5.3.2). As a first step, let us analyze the bounds on the execution of higher-priority tasks, the source of interference.

Execution of higher-priority tasks Assuming a time window of size t, let $X_{i,j}$ be the cumulative execution time of a higher-priority task T_i on some processor Π_j in such an interval while T_k is not executing. Recall from Section 5.1 that $H_k^i(t)$ denotes an upper bound on the interference incurred by task T_k due to the higher-priority task T_i. Given that T_i's execution does not exceed its total interference on T_k, and that affinity restrictions must be respected, the execution of higher-priority tasks is bounded according to the following lemma.

Lemma 8 *In any schedule of τ:*

$$\forall T_i \in hp_k : \sum_{\Pi_p \in \alpha_i} X_{i,p} \leq H_k^i(t). \tag{18}$$

Proof Consider any higher-priority task T_i. By definition, $X_{i,p}$ denotes the total execution time of T_i on processor Π_p while T_k is not executing. Thus, the accumulated execution of T_i on all the processors in α_k cannot be larger than the upper bound $H_i^k(t)$ on the interference incurred by T_k. □

The fact that tasks can be confined to execute only on certain processors also leads to restrictions on the execution time.

Lemma 9 *In any schedule of τ:*

$$\forall T_i \in hp_k, \forall \Pi_p \notin \alpha_i : X_{i,p} = 0. \tag{19}$$

Proof Follows trivially from the fact that T_i cannot be scheduled on processors that are not part of its processor affinity set α_i. □

Finally, given the bounds on execution time, it is possible to determine on a per-processor granularity how much interference can be incurred by a task T_k in the worst case.

Per-processor interference Using the per-processor execution constraints for high-priority tasks, let us infer for how long T_k is able to execute in an interval of size t. For that, consider the total execution time of higher-priority tasks on each processor $\Pi_p \in \alpha_k$, which we denote by $\sum_{T_i \in hp_k} X_{i,p}$. In any possible schedule, the processor that minimizes this term bounds the response time of T_k. This follows from the fact that, for every processor Π_p, the interference incurred by T_k on Π_p cannot be larger than the total execution time of higher-priority tasks on Π_p. This implies the following invariant.

Lemma 10 *Let x be the largest value such that*

$$\forall \Pi_p \in \alpha_k : x \leq e_k + \sum_{T_i \in hp_k} X_{i,p}.$$

Then, in any schedule of τ, x is an upper-bound on the response time of task T_k.

Proof Suppose not. Then there exists a job of T_k in some legal schedule whose response time r_k satisfies:

$$\exists \Pi_p \in \alpha_k : r_k > e_k + \sum_{T_i \in hp_k} X_{i,p}.$$

This implies that a job completes $e_k + \sum_{T_i \in hp_k} X_{i,p} + \Delta$ time units after its arrival, with $\Delta > 0$. Since T_k can only be delayed by the interference of higher-priority tasks, it can only be the case that Π_p stays idle for Δ time units while task T_k is backlogged. This is impossible under a work-conserving scheduler. □

As we show next, the three preceding lemmas can be combined in a simple way to construct an LP that yields an upper bound on the response time.

5.3.2 LP Formulation

The formulation of the LP follows straightforwardly from Lemmas 8–10. All LP variables are denoted by uppercase letters to avoid ambiguity.

Consider the execution of any job of some task T_k. In order to satisfy the temporal constraints, deadlines must be met in the presence of the maximum possible interference by higher-priority tasks. Therefore, we propose an LP that **maximizes the response-time bound $\mathbf{R_k}$**, constrained by the limits on interference. Though our model assumes integer time, integer programming is not required as we show in Section 5.3.3. Thus, all LP variables are real-valued.

The variable R_k represents the response-time bound for any job of a task T_k in any time window of size t starting with the job arrival. In the LP, apart from R_k, there are $m \cdot |hp_k|$ variables $X_{i,p}$, which express the interference of each higher-priority task on each processor that is part of T_k's processor affinity. Let $R_k^{LP}(t)$ denote the solution R_k of the LP for a given parameter t. The LP is defined as follows:

$$R_k^{LP}(t) \triangleq \text{maximize } \; R_k \; \text{ subject to}$$

$$\forall T_i \in hp_k : \sum_{\Pi_p \in \alpha_i} X_{i,p} \leq H_k^i(t) \qquad \text{(Constraint 1)}$$

$$\forall T_i \in hp_k, \forall \Pi_p \notin \alpha_i : X_{i,p} = 0 \qquad \text{(Constraint 2)}$$

$$\forall \Pi_p \in \alpha_k : R_k \leq e_k + \sum_{T_i \in hp_k} X_{i,p} \qquad \text{(Constraint 3)}$$

The validity of the constraints in the definition of $R_k^{LP}(t)$ is a consequence of Lemmas 8–10. Constraints 1 and 2 follow directly from Lemmas 8 and 9, since they represent the bounds on the execution of higher-priority tasks on each processor. Because of the maximization of the objective function, instantiating $x = R_k$ in Lemma 10 guarantees that R_k is a valid response-time bound for T_k.

This LP only allows computing the response-time of task T_k in a restricted time window. For deriving a schedulability analysis, we need to apply a fixed-point iteration on the size of the interval, as discussed next.

5.3.3 LP-based RTA

In order to obtain an upper bound on the response time for any interval, we must apply a fixed-point iteration based on Equation 16. In each iteration, a new response-time bound r_k^{ub} is calculated by solving the LP, as the workload of higher-priority tasks grows with the analyzed interval, until convergence. Thus, the response time can be computed with the following fixed-point iteration (starting with $r_k^{ub} = e_k$):

$$r_k^{ub} \leftarrow R_k^{LP}(r_k^{ub}). \tag{20}$$

Since the system model assumes integer time (*e.g.*, processor cycles), it is sufficient to round down each computed value of $R_k^{LP}(r_k^{ub})$ instead of representing execution with integer variables. Because the objective function is maximized, the solution of the LP-relaxation of the problem provides a valid upper bound. Let R_k^{RND} denote the rounded integer solution:

$$R_k^{RND}(r_k^{ub}) \triangleq \lfloor R_k^{LP}(r_k^{ub}) \rfloor. \tag{21}$$

Then the fixed-point iteration can be defined as follows:

$$r_k^{ub} \leftarrow R_k^{RND}(r_k^{ub}). \tag{22}$$

In the theorem below, we formally state the correctness of the LP-based approach.

Theorem 6 The response time of task T_k is upper-bounded by the value of r_k^{ub} obtained via the LP-based fixed-point iteration in Equation 22.

Proof Let r_k be the maximum response time of task T_k in any legal schedule. Consider the interference of higher-priority tasks in a time window of size r_k, starting with the arrival of a job that incurs the maximum response time. Assuming convergence, let $r_k^{ub} \leq D_k$ be the fixed-point of the iteration defined in Equation 22. We must prove that $r_k \leq r_k^{ub}$.

As discussed, $r_k^{ub} = \lfloor R_k^{LP}(r_k^{ub}) \rfloor$ is the integer solution of the LP-relaxation. By the definition of the floor function, the real-valued solution $R_k = R_k^{LP}(r_k^{ub})$ satisfies $R_k < r_k^{ub} + 1$. Therefore, it suffices to show that $r_k \leq R_k$.

The values of $X_{i,p}$ and R_k in the solution of the LP satisfy all the Constraints 1-3. The execution of higher-priority tasks is bounded by Constraints 1 and 2, which were proven, in Lemmas 8 and 9, to be true for every schedule. Further, since the objective function is maximized, R_k is the largest value that preserves Constraint 3. Therefore, according to Lemma 10, R_k is an upper bound on the maximum response time r_k, *i.e.*, $r_k \leq R_k$. □

Having shown that the analysis provides a correct upper bound on the maximum response time, we now prove that the LP-based analysis does not perform worse than the RTA analysis presented in Section 5.2. First, let us define $R_k^{sub}(t)$ as the RHS of Equation 17:

$$R_k^{sub}(t) \triangleq e_k + \min_{\substack{\forall s \subseteq \alpha_k \\ s \neq \emptyset}} \left\lfloor \frac{1}{|s|} \cdot \sum_{\forall T_i \in hp_k : \alpha_i \cap s \neq \emptyset} H_k^i(t) \right\rfloor. \tag{23}$$

In order to compare the fixed-points obtained via Equations 17 and 22, we first show that the function R_k^{RND} is dominated by R_k^{sub}.

Lemma 11 For any task T_k and any time window of size t, $R_k^{RND}(t) \leq R_k^{sub}(t)$.

Proof Let $R_k = R_k^{LP}(t)$ be the solution of the LP assuming a time window of size t, where $t \geq 0$. Let $s \subseteq \alpha_k$ be any non-empty subset of α_k. It follows that:

$$R_k = \frac{R_k \cdot |s|}{|s|} \tag{24}$$

$\{$Rewrite $R_k \cdot |s|$ as a sum.$\}$

$$= \frac{\sum_{\Pi_p \in s} R_k}{|s|} \tag{25}$$

$\{$For each processor $\Pi_p \in s$, substitute R_k according to Constraint 3.$\}$

$$\leq \frac{\sum_{\Pi_p \in s} \left(e_k + \sum_{T_i \in hp_k} X_{i,p} \right)}{|s|} \tag{26}$$

$\{$Extract e_k, divide by $|s|$.$\}$

$$= e_k + \frac{1}{|s|} \cdot \sum_{\Pi_p \in s} \left(\sum_{T_i \in hp_k} X_{i,p} \right) \tag{27}$$

$\{$Transpose the indices of the nested summation.$\}$

$$= e_k + \frac{1}{|s|} \cdot \sum_{T_i \in hp_k} \left(\sum_{\Pi_p \in s} X_{i,p} \right) \tag{28}$$

$\{$Split the outer sum into two mutually exclusive cases, $\alpha_i \cap s \neq \emptyset$ and $\alpha_i \cap s = \emptyset$.$\}$

$$= e_k + \frac{1}{|s|} \cdot \left(\sum_{\substack{T_i \in hp_k \\ \alpha_i \cap s \neq \emptyset}} \sum_{\Pi_p \in s} X_{i,p} + \sum_{\substack{T_i \in hp_k \\ \alpha_i \cap s = \emptyset}} \sum_{\Pi_p \in s} X_{i,p} \right) \tag{29}$$

$\{$From Constraint 2, for every $T_i \in hp_k$ such that $\alpha_i \cap s = \emptyset$, $\forall \Pi_p \in s : X_{i,p} = 0$. $\}$

$$= e_k + \frac{1}{|s|} \cdot \sum_{\substack{T_i \in hp_k \\ \alpha_i \cap s \neq \emptyset}} \sum_{\Pi_p \in s} X_{i,p} \tag{30}$$

{The total execution of T_i on the processors in s, given by $\sum_{\Pi_p \in s} X_{i,p}$, cannot be greater than the upper bound $H_k^i(t)$ on the total interference caused by task T_i.}

$$\leq e_k + \frac{1}{|s|} \cdot \sum_{\substack{T_i \in hp_k \\ \alpha_i \cap s \neq \emptyset}} H_k^i(t). \tag{31}$$

Since s denotes any non-empty subset of α_k, the inequality also holds for the particular subset that minimizes the RHS of the inequality.

$$R_k \leq e_k + \min_{\substack{\forall s \subseteq \alpha_k \\ s \neq \emptyset}} \left\{ \frac{1}{|s|} \cdot \sum_{\substack{T_i \in hp_k \\ \alpha_i \cap s \neq \emptyset}} H_k^i(t) \right\} \tag{32}$$

By rounding-down both sides of the inequality (since $a \leq b$ implies $\lfloor a \rfloor \leq \lfloor b \rfloor$ for any two reals a, b), we obtain

$$R_k^{RND}(t) = \left\lfloor R_k \right\rfloor \leq \min_{\substack{\forall s \subseteq \alpha_k \\ s \neq \emptyset}} \left\lfloor \frac{1}{|s|} \cdot \sum_{\substack{T_i \in hp_k \\ \alpha_i \cap s \neq \emptyset}} H_k^i(t) \right\rfloor = R_k^{sub}(t). \tag{33}$$

□

Note that for any non-empty subset $s \subseteq \alpha_k$, Equation 27 could be easily encoded in the LP. However, this is not needed, since this property is already implied by the LP constraints, and therefore implicitly assumed in the solution space via linear dependence.

Next, we compare the fixed-points of the two proposed RTAs. In the following, we let $f^i(0)$ denote the repeated iterative application of f to the starting value 0; formally, we define a fixed-point iteration for some function f starting at 0 recursively as $f^0(0) = f(0)$ and $f^{i+1}(0) = f(f^i(0))$, for $i \geq 0$. In particular, the two functions representing the LP- and reduction-based response-time bounds have the starting points at $R_k^{LP}(0) = R_k^{sub}(0) = e_k$.

Next, we show that dominance relation among R_k^{LP} and R_k^{sub} established in Lemma 11 also implies dominance among the respective fixed-points, as shown next.

Lemma 12 Let $f(t)$ and $g(t)$ be two monotonically increasing functions on the natural numbers. *Assume that $f(t)$ and $g(t)$ have least fixed-points $\mu f = f^n(0)$ and $\mu g = g^m(0)$, with n and m finite. If $\forall t : f(t) \leq g(t)$, then $\mu f \leq \mu g$.*

Proof By contrapositive, assume $\mu g < \mu f$. Consider the fixed-point iterations of f and g, starting at 0, each taking a minimum of n and m steps for convergence, respectively.

$$< 0, f^1(0), f^2(0), \ldots, f^n(0) = \mu f, \ldots >$$

$$< 0, g^1(0), g^2(0), \ldots, g^m(0) = \mu g, \ldots >$$

Since f is monotonically increasing, it can be proven by induction that $\forall i < n : f^i(0) \leq f^{i+1}(0)$. Further, because μf is the least fixed point of f, $\forall i < n : f^i(0) < f^{i+1}(0)$.

From the assumption that $\mu g = g^m(0) < \mu f$, it follows that there exists an i, where $0 \leq i \leq n$, such that:

$$0 < \ldots < f^{n-i-1}(0) \leq g^m(0) < f^{n-i}(0) < \ldots < f^n(0).$$

This implies that:

$$f^{n-i-1}(0) \leq g^m(0).$$

Since f is monotonic, we can apply f to both sides of the inequality:

$$f(f^{n-i-1}(0)) \leq f(g^m(0)).$$

By definition of f^n:

$$f^{n-i}(0) \leq f(g^m(0)).$$

Since $g^m(0) < f^{n-i}(0)$:

$$g^m(0) < f^{n-i}(0) \leq f(g^m(0)).$$

Since by assumption $g^m(0)$ is a fixed point of g:

$$g(g^m(0)) = g^m(0) < f^{n-i}(0) \leq f(g^m(0)).$$

By choosing $t = g^m(0)$, this shows that $\exists t : f(t) > g(t)$, proving the contrapositive. □

Using Lemmas 11 and 12, we can now prove that the LP-based analysis never performs worse than the analysis based on the reduction to subproblems.

Theorem 7 The response-time bound r_k^{ub} of a task T_k, obtained via the LP-based fixed-point iteration (Equation 22), is less than or equal to the response-time bound computed via reduction to subproblems (Equation 17).

Proof From Lemma 11, it follows that $\forall t : R_k^{LP}(t) \leq R_k^{sub}(t)$. Thus, according to Lemma 12, after applying the fixed-point iteration, the response-time obtained with $R_k^{sub}(t)$ cannot be less than the response-time obtained with $R_k^{LP}(t)$. This shows that the LP-based analysis dominates the response-time analysis based on reduction to subproblems. □

An important outcome of the LP-based approach is the reduced computational complexity when analyzing subsets of processor affinities. This leads to a more efficient way to perform the response-time analysis than the approach based on the reduction to“global-like” subproblems. For each step of the iteration, the solution can be computed solving an LP, instead of evaluating the interference for each subset of α_k. This effectively reduces the complexity of each iteration to polynomial time. In practice, it also allows employing optimized, off-the-shelf LP-solvers to achieve good performance.

This concludes our discussion of response-time analysis for APA scheduling with fixed priorities. Next, we report on an empirical evaluation of the various schedulability analysis approaches introduced in this paper.

6 Experiments and Evaluation

In this section, we present the results of two sets of experiments we performed to evaluate different aspects of APA scheduling. In particular, we sought to assess whether it provides schedulability gains over global and partitioned scheduling, whether the proposed analyses induce significant pessimism, and whether the proposed analyses are scalable w.r.t. the size of the problem, *i.e.*, the number of processors m and the number of tasks n.

In the first set of experiments, we focused on the proposed reduction-based analyses, *i.e.*, the exhaustive approach and the heuristic-based approach proposed in Sections 4.2 and 4.3. We first compared APA scheduling with global and partitioned scheduling in terms of schedulability using randomly-generated tasksets. Second, we compared the two reduction-based approaches with each other to assess if the heuristic is sufficiently accurate in identifying promising subproblems to avoid excessive pessimism.

In the second set of experiments, we shifted our focus to the LP-based schedulability analysis. Since the LP-based approach has lower computational complexity and performs as well as the exhaustive-reduction-based approach (*i.e.*, all tasksets claimed to be schedulable by exhaustive-reduction-based analysis are also claimed to be schedulable by the LP-based analysis), this allows us to evaluate a larger range of processor counts. We further compared the LP-based analysis with upper bounds on schedulability derived using a feasibility test and an un-schedulability test (explained in Section 6.3) in order to achieve a better notion of how the proposed APA schedulability analyses perform.

We next describe the experimental setup and then report on the observed trends.

6.1 Experimental Setup

In our experiments, tasksets were generated using two different methods. The first set of experiments, which evaluated the proposed reduction-based approaches, used the taskset generator designed by Emberson et al (2010), whereas the second experiment, which targets the LP-based approach, included tasksets generated similarly to previous LITMUS^{RT} studies (Brandenburg, 2011). The exact taskset generation parameters for an experiment are described along with the respective experiment.

We assume implicit deadlines in our experiment because **(i)** the only available feasibility test for APA scheduling (Baruah and Brandenburg, 2013) applies only to implicit-deadline tasks; **(ii)** when assessing the scalability of the reduction-based methods, the choice of constrained or implicit deadlines is irrelevant; and **(iii)** tasksets with constrained deadlines are more difficult to schedule and hence would require more involved affinity assignment heuristics, which are not the focus of this work.

Because of its widespread use, as a first step in evaluating APA scheduling, we restricted our focus to fixed priority policies. For global FP scheduling, we used the response-time analysis for global fixed-priority scheduling given by Bertogna and Cirinei (2007), which we denote as G-FP-RTA. For partitioned FP scheduling (P-FP), we used uniprocessor response-time time analysis (Audsley et al, 1993) and assigned tasks to processors in order of decreasing utilization. In all cases, task priorities were assigned according to the DkC heuristic, which reduces to assigning deadline-monotonic priorities in the partitioned case (Davis and Burns, 2011a).

To partition tasksets, we used five standard bin-packing heuristics: *worst-fit-decreasing*, *first-fit-decreasing*, *best-fit decreasing*, *next-fit-decreasing*, and *almost-worst-fit-decreasing*.

A taskset 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 reduce pessimism in our experiments, in the response-time analysis for APA scheduling we included an optimization for uniprocessors affinity masks, which is explained next.

6.1.1 Optimization for the Uniprocessor Case

Recall from Section 4.2 that when analyzing the interference on task T_k due to higher-priority tasks using the reduction-based approach, we search for the subset $s \subseteq \alpha_k$ that results in the least interference. An opportunity for reducing analysis pessimism arises when considering singleton subsets, which arise either when $|s| = 1$ or when $|\alpha_k| = 1$ (*i.e.*, either T_k does not migrate because of the affinity restrictions, or because we assume that as an analysis argument). In both cases we can consider that the analyzed task T_k and all the higher-priority tasks in hp_k execute entirely on the single processor in s and do not migrate. This allows applying the exact response-time analysis for uniprocessor systems, which does not incur the extra pessimism inherent in current global schedulability tests.

Thus, whenever $|s| = 1$, we can replace the workload $H_k^i(t)$ of a higher-priority task T_i with the less pessimistic interference $H_k^{i,UNI}(t)$ for uniprocessor systems, defined as follows.

$$H_k^{i,UNI}(t) = \left\lceil \frac{t}{p_i} \right\rceil \cdot e_i \tag{34}$$

The RTA test based on Equation 17 can be adapted to use the uniprocessor test as follows, by analyzing the different subsets separately:

$$r_k^{ub} \leftarrow e_k + \min\left\{ \min_{\substack{s\subseteq\alpha_k \\ |s|>1}} \left\lfloor \frac{1}{|s|} \cdot \sum_{\substack{T_i\in hp_k \\ \alpha_i\cap s\neq\emptyset}} H_k^i(r_k^{ub}) \right\rfloor,\ \min_{\substack{s\subseteq\alpha_k \\ |s|=1}} \left\lfloor \frac{1}{|s|} \cdot \sum_{\substack{T_i\in hp_k \\ \alpha_i\cap s\neq\emptyset}} H_k^{i,UNI}(r_k^{ub}) \right\rfloor \right\}. \tag{35}$$

The LP can similarly be extended with the following set of per-processor constraints:

$$\forall \Pi_p \in \alpha_k : R_k \leq e_k + \sum_{\substack{T_i\in hp_k \\ \Pi_p\in\alpha_i}} H_k^{i,UNI}(t). \qquad \text{(Constraint 4)}$$

Each constraint limits the response-time on processor $\Pi_p \in \alpha_k$ assuming that every interfering task T_i such that $\Pi_p \in \alpha_i$ executes only on Π_p for the entire interval. That is, when analyzing task T_k, Π_p is assumed to be the only processor in the system. The extra $|\alpha_k|$ constraints bound R_k by a constant value and thus do not affect Constraints 1–3. In our experiments, we used the refined interference bounds to achieve less pessimistic results, which had a significant impact especially when comparing APA and partitioned scheduling.

Having discussed the experimental setup, we next discuss each experiment and the corresponding results.

6.2 Reduction-based Schedulability Analysis

The experimental results in this section demonstrate that the proposed generic analysis—reduction to "global-like" subproblems—is indeed effective. While Experiment 1 compared global, partitioned, and APA FP scheduling, Experiment 2 assessed the performance of the proposed heuristic-based schedulability analysis w.r.t. the exhaustive approach. Further comparisons of APA scheduling with other scheduling techniques and deriving other affinity mask assignment heuristics for evaluation would certainly be interesting; however, such studies are beyond the scope of this paper and remain the subject of future work.

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 tasksets used in the first experiment were assigned processor affinities using a heuristic similar to the one discussed in Section 4, *i.e.*, we started with a global assignment and allowed shrinking of the processor affinities until a schedulable partitioned assignment was found, or until a schedulable arbitrary assignment (*i.e.*, an intermediate assignment in between global and partitioned assignments) was achieved. Since optimal priority assignment for APA scheduling is still an open problem, we used the DkC priority assignment with the aforementioned heuristic (Davis and Burns, 2011a). As mentioned in Section 6.1.1, tasks with a singleton processor affinity set were analyzed using uniprocessor response time analysis (instead of G-FP-RTA) for improved accuracy.

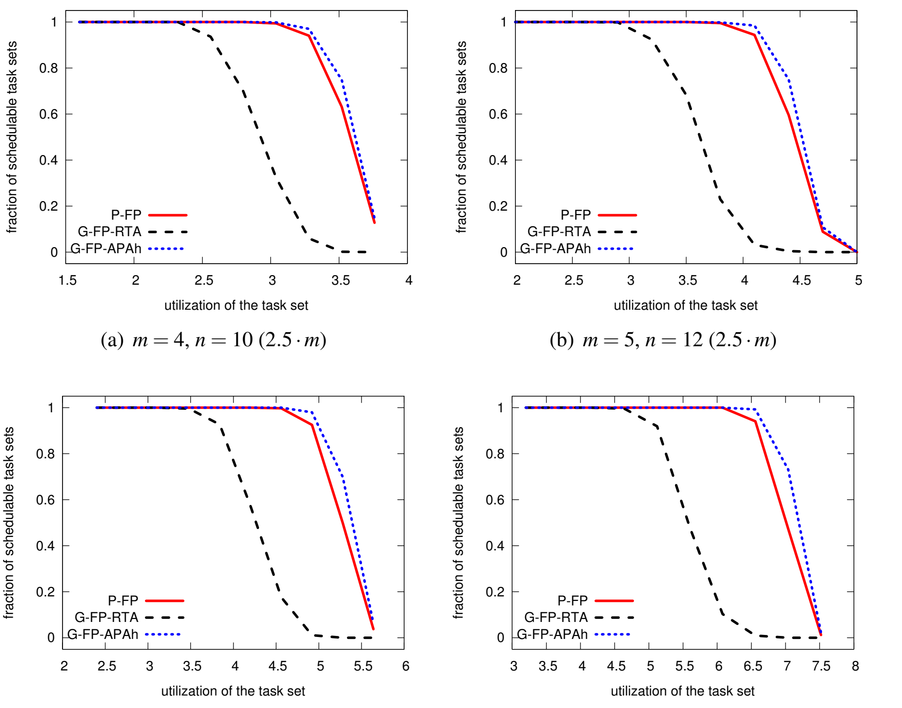
We considered two variants of G-FP-APA, the exhaustive approach based on Theorem 3 (G-FP-APAe) and the heuristic-based approach based on Algorithm 1 (G-FP-APAh). For Experiment 1, we varied the number of processors m from 3 to 8. Herein, we focus on graphs corresponding to $m \in \{4, \ldots, 8\}$. For Experiment 2, m ranged 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 tasksets, with a number of tasks ranging in $\{m+1, 1.5m, 2m, 2.5m\}$, to allow for tasksets that are not easily partitionable. The periods of tasks were randomly chosen from [10ms, 100ms] following a log-uniform distribution. We summarize the main trends apparent in the results of Experiments 1 and 2 below.

Experiment 1 (G-FP-APAh 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 tasksets schedulable as a function of the total system utilization. For utilizations greater than 75%, G-FP-APAh 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 tasksets that cannot be partitioned easily, nor are schedulable by global scheduling. However, in the experiment, G-FP-APAh does not exhibit a large improvement over its partitioned and global counterparts for the tested workloads, because the generated tasksets were not very challenging for the global and partitioned schedulers.

There are two causes for this effect. First, (randomly) generating such tasksets without biasing towards a specific configuration is a challenging problem in itself, and second, the pessimism in global schedulability analysis (which is also inherited by APA scheduling) limits the number of tasksets with high utilization that are schedulable. Another reason for such a small improvement in schedulability is that determining (provably) good combinations of priority and affinity mask assignments remains an open problem, which bottlenecks the schedulability of many workloads (that otherwise might have been schedulable).

Overall, the results shown in Figure 3 demonstrate that APA scheduling has the potential to improve schedulability, but also indicate that substantial further work is required to fully exploit the potential.

Experiment 2 (G-FP-APAh vs. G-FP-APAe) The objective of this experiment was 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 number



(a) $m = 4$, $n = 10$ $(2.5 \cdot m)$

(b) $m = 5$, $n = 12$ $(2.5 \cdot m)$

(c) $m = 6$, $n = 15$ $(2.5 \cdot m)$

(d) $m = 8$, $n = 20$ $(2.5 \cdot m)$

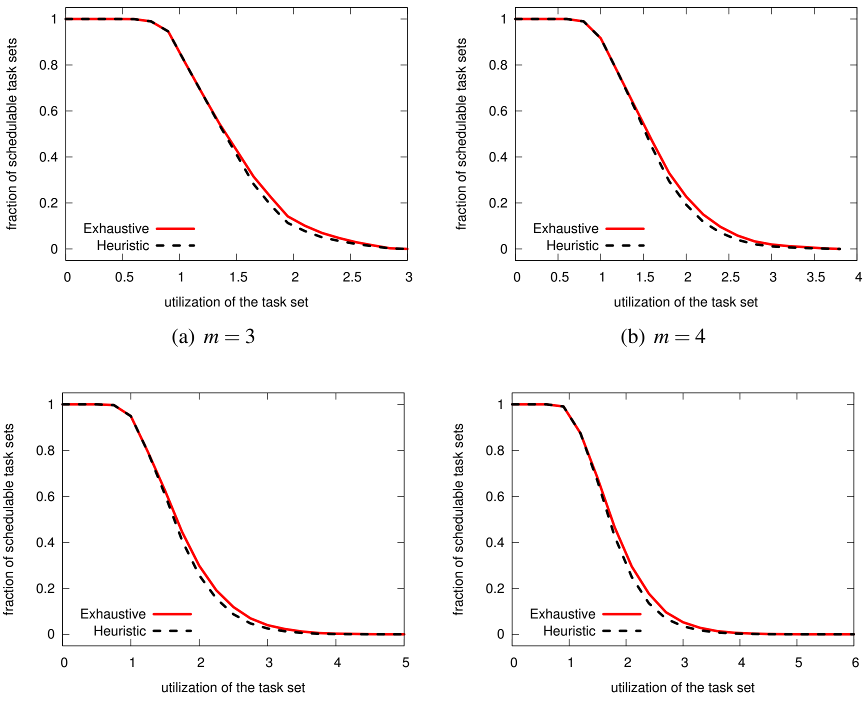
Fig. 3 The comparison of APA scheduling (G-FP-APAh) versus global (G-FP-RTA) and partitioned (P-FP) scheduling.

of processors was reduced because in our test environment the exhaustive test did not scale beyond $m = 5$. The results in Figure 4 show that G-FP-APAh performs almost as well as G-FP-APAe, *i.e.*, the curves diverge only slightly. This validates the efficiency of the proposed heuristic. Note that processor affinities were generated randomly in this experiment, which explains the overall lower schedulability compared to Experiment 1.

Overall the, results of Experiments 1 and 2 demonstrate that the proposed analysis—reduction to global subproblems—is indeed effective, though its potential is currently limited by the absence of exact JLFP-scheduling analysis and systematic ways of choosing priorities and affinities due to the combinatorial explosion (*i.e.*, n tasks executing on m processors can have up to $2^{m \cdot n}$ different affinity assignments). Further comparisons of APA scheduling with other scheduling techniques and the development of improved affinity assignment algorithms are left as future work.

6.3 LP-based Schedulability Analysis

Recall from Section 5.3 that LP-based RTA for FP scheduling is provably as good as the reduction-based analysis with exhaustive reductions, while having a much lower per-iteration complexity. And in practice, our implementation indeed performs much faster in comparison (*i.e.*, the overheads of invoking an LP solver are outweighed by the efficiency improvements).



(a) $m = 3$ (b) $m = 4$ (c) $m = 5$ (d) $m = 6$

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

To empirically evaluate how well the LP-based analysis scales, Experiment 3 performs schedulability experiments for a number of processors ranging up to 32. In the following, we denote the LP-based schedulability analysis as G-FP-APA-LP.

Furthermore, we assessed how the proposed schedulability analysis for APA scheduling performs in general, by plotting curves for two upper bounds on schedulability based on a feasibility test and a simulation-based "un-schedulability" test, as explained next.

Feasibility test for JLDP APA scheduling Baruah and Brandenburg (2013) recently proposed a feasibility test based on the connection between APA scheduling and scheduling on unrelated heterogenous multiprocessors. The test assumes implicit deadlines and also employs linear programming. Unlike the schedulability analyses proposed in this paper, the feasibility test assumes JLDP scheduling, which strictly generalizes scheduling with FP and JLFP policies. Though many of the tasksets reported as feasible are not representative of JLFP policies, the curve serves as an upper bound for APA scheduling in general. Since tasksets that are not schedulable under JLDP policies are also infeasible under JLFP policies, this provides an indication of parameter choices that lead to infeasible tasksets. We refer to this test as JLDP-APA-feas.

"Un-schedulability" test for FP APA scheduling A feasibility curve for JLDP policies in itself may be misleading given that the rest of the analyses evaluated correspond to

FP scheduling and assume tasks are prioritized according to DkC. Therefore, we added a simple “un-schedulability” test, obtained by simulating tasksets under APA scheduling with synchronous, periodic arrivals for a thousand seconds. While such a simulation cannot be used to establish that a given taskset is schedulable, observing a deadline miss does indicate that it is certainly not schedulable. The simulation-based “un-schedulability” test hence provides an upper bound on the schedulability that could be achieved by an exact schedulability test. We include it here to provide a context for the accuracy of the proposed sufficient, but not necessary, LP-based RTA. We denote the “un-schedulability” test as FP-APA-sim.

While the feasibility test results are irrespective of task priorities, the simulation and schedulability tests are, however, based on a particular priority assignment (DkC). Therefore, finding better strategies to assign priorities could potentially improve schedulability, bridging the gap between the feasibility test and the remaining curves.

We next briefly detail the experimental setup and then report on the observed results.

Experiment 3 (G-FP-APA-LP vs. JLDP-APA-feas vs. FP-APA-sim vs. G-FP-APAh) Unlike in Experiments 1 and 2, in this experiment we used two taskset generators. In addition to tasksets with uniformly distributed utilization generated with Emberson *et al.*’s method (2010), we also tested tasksets obtained similarly to previous LITMUS$^{\mathrm{RT}}$ experiments (*e.g.*, see Bastoni et al, 2011; Brandenburg, 2011). The second method assumed a bimodal distribution, with utilizations ranging uniformly over [0.001, 0.5) and [0.5, 0.9], with probabilities of 4/9 and 5/9, respectively. This distribution favors tasksets of larger utilization variance, which are in general more difficult to schedule and led to contrasting results in our experiments. We refer to the two taskset distributions as *uniform* and *bimodal heavy*, respectively.

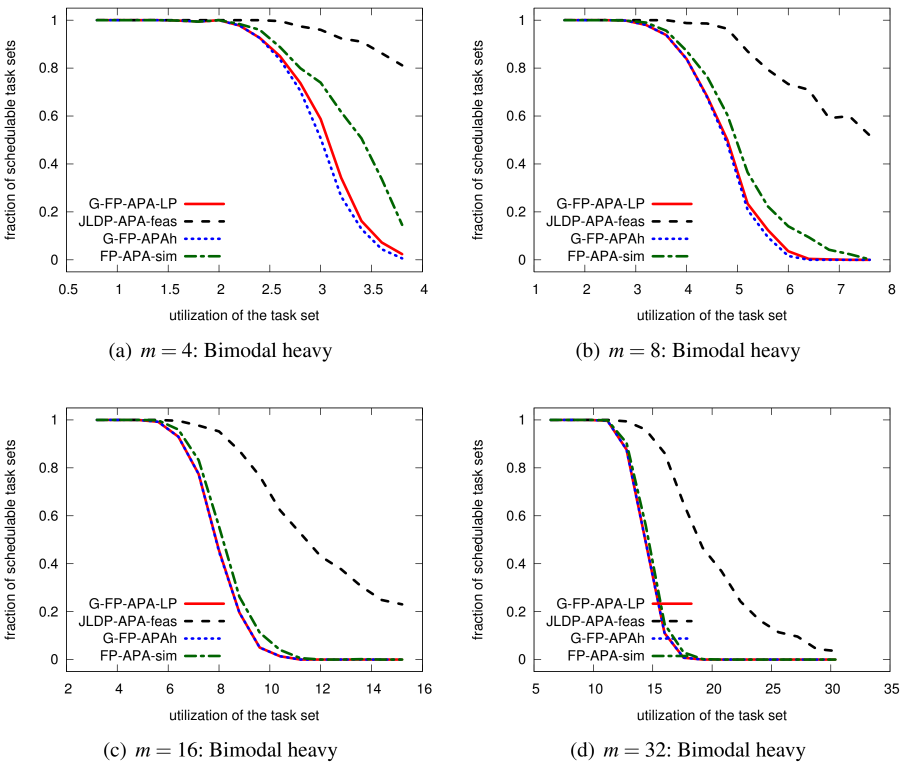
We considered two variants of G-FP-APA, the LP-based approach (G-FP-APA-LP) and the heuristic-based approach (G-FP-APAh). The number of processors was varied across $m \in \{4, 8, 16, 32\}$.

For the tasksets generated with the *uniform* distribution, we let the number of tasks n range over $\{m+1, 2m, 3m, 4m, 5m, 6m\}$. For every graph corresponding to m processors and n tasks, we varied the utilization cap on the x-axis from 0 to m in steps of 0.5, with each point representing 500 sample tasksets. The periods of tasks were randomly chosen from [10ms, 100ms] from a log-uniform distribution.

For the tasksets generated with the *bimodal heavy* distribution, the number of tasks is implicitly determined based on the specified target utilization (*i.e.*, tasks are added to the taskset until adding another task would exceed the desired target utilization). For every graph corresponding to m processors, we varied only the utilization cap on the x-axis from 0 to m in steps of 0.5. Each point in the graph represents 500 sample tasksets, and the periods of tasks were randomly chosen from [10ms, 100ms] from a uniform distribution.

Instead of generating random affinity assignments as in Experiment 2, we adopted a different approach to avoid generating primarily infeasible tasksets. To this end, we used the following heuristic.

Assume a system with $m = 2^k$ processors, for some $k \geq 0$. We begin by assigning the m highest-priority tasks to individual processors. At step 1, we assign the next $m/2$ tasks to each pair of 2 processors. At step 2, the next $m/4$ tasks to each group of 4 processors, and so on. In case tasks remain, they are assigned global affinities. This leads to a hierarchical affinity assignment, where the highest-priority tasks execute on smaller affinity sets. As we see in Figures 5 and 6, the overall schedulability is significantly higher than in Experiment 2. In contrast, when using random affinity assignments, the overall schedulability decreased already with utilizations as low as 25 percent of the total system capacity for 5 processor systems



(a) $m = 4$: Bimodal heavy
(b) $m = 8$: Bimodal heavy
(c) $m = 16$: Bimodal heavy
(d) $m = 32$: Bimodal heavy

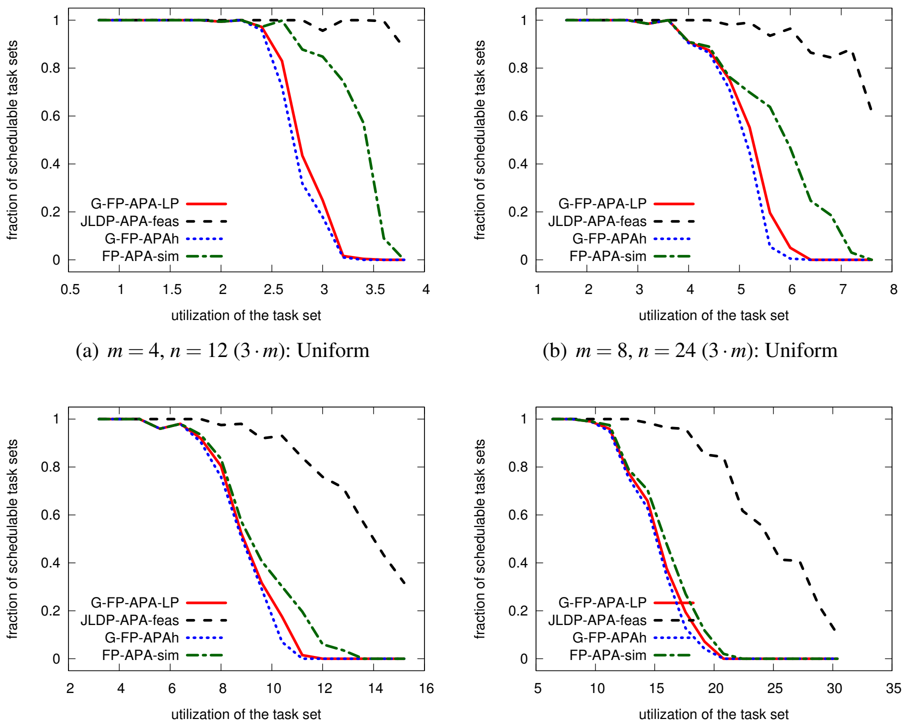
Fig. 5 The comparison of the LP-based Analysis (G-FP-APA-LP) with the APA Feasibility Test (JLDP-APA-feas), the heuristic-based reduction (G-FP-APAh) and the APA simulation (FP-APA-sim), for tasksets generated using the bimodal heavy distribution.

(recall Figure 4). This follows from the fact that, with random affinities, processors are more likely to become overloaded, since tasks are allocated with no load-balancing strategy.

The resulting graphs are shown in Figures 5–7. First of all, the most reassuring result from Experiment 3 is the fact that the LP-based approach scales well with the magnitude of the parameters n and m, compared with the exhaustive approach shown in Experiment 2. With respect to the computational cost of the analysis, G-FP-APA-LP outperformed G-FP-APAe in our implementation, making accurate APA schedulability analysis viable for larger use cases as well.

Regarding the schedulability results, we can see that the taskset parameters heavily influence the results of a particular schedulability test, so properties of different taskset distributions should not be generalized. In the case of bimodal heavy tasksets the schedulability analysis and simulation results are similar (see Figure 5), because the affinity assignment may be too restrictive for some of the heavy tasks (which occur more frequently in the bimodal distribution), and because tasksets with large utilization variance are generally difficult to schedule on multiprocessors.

However, when confronted with uniform tasksets, the analysis techniques perform worse than the simulation, especially in scenarios with a small number of processors and high total utilization (see Figure 6). This is caused by a combination of two factors: the pessimism in the



(a) $m = 4$, $n = 12$ $(3 \cdot m)$: Uniform

(b) $m = 8$, $n = 24$ $(3 \cdot m)$: Uniform

(c) $m = 16$, $n = 48$ $(3 \cdot m)$: Uniform

(d) $m = 32$, $n = 96$ $(3 \cdot m)$: Uniform

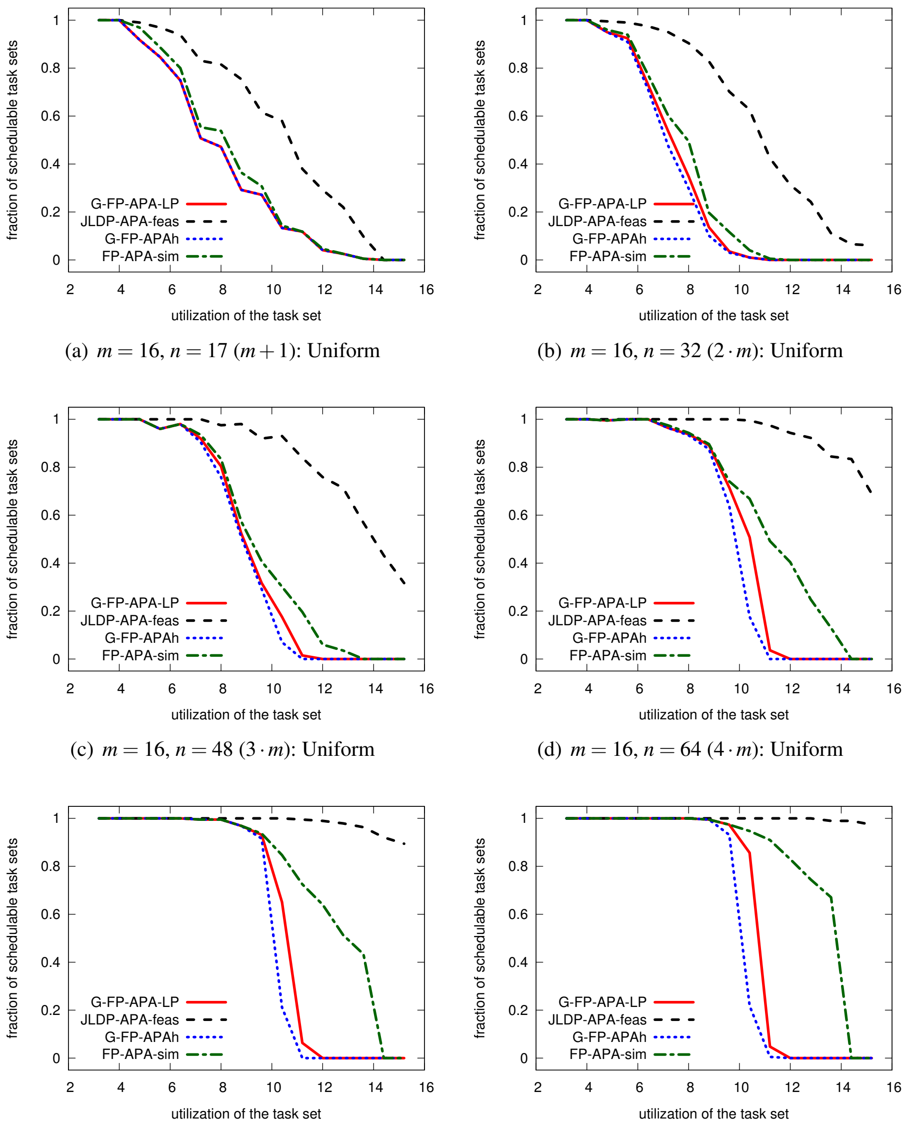
Fig. 6 The comparison of the LP-based Analysis (G-FP-APA-LP) with the APA Feasibility Test (JLDP-APA-feas), the heuristic-based reduction (G-FP-APAh) and the APA simulation (FP-APA-sim), for tasksets generated using uniform distribution with a fixed tasks-per-processor ratio of 3.

analysis of interference, which lowers the schedulability curves, and the fact that the simulated arrival sequence may not represent the worst case, which causes the un-schedulability bound to not be tight.

Also, in both Figures 5 and 6, schedulability starts to decrease at (relatively) smaller utilizations as the number of processors increases. When compared with the feasibility curve, this shows that for large processor counts there is not much room for improvement in the results, except by changing the scheduling policy (*e.g.*, using a JLDP policy may improve schedulability) or by devising better affinity assignment heuristics.

Figure 7 shows graphs from the same experiment for the *uniform* distribution, but with a fixed number of processors equal to 16. As we can see, with an increasing number of tasks, the gap between analysis and simulation also increases. This is due to the pessimistic carry-in bound in the multiprocessor response-time analysis, which is $O(n)$. However, incorporating Baruah (2007)'s technique to limit the carry-in work to $O(m)$, *i.e.*, to analyze scenarios in which carry-in interference arises due to at most m other tasks, could help to narrow this gap.

Also note that though it is hard to establish a connection between the priority ordering, affinity assignment, and task distribution, the feasibility test provides some insights as it does not consider priorities. The trends w.r.t. the feasibility test in Figure 7 show that increasing



(a) $m = 16$, $n = 17$ $(m+1)$: Uniform

(b) $m = 16$, $n = 32$ $(2 \cdot m)$: Uniform

(c) $m = 16$, $n = 48$ $(3 \cdot m)$: Uniform

(d) $m = 16$, $n = 64$ $(4 \cdot m)$: Uniform

(e) $m = 16$, $n = 80$ $(5 \cdot m)$: Uniform

(f) $m = 16$, $n = 96$ $(6 \cdot m)$: Uniform

Fig. 7 The comparison of the LP-based Analysis (G-FP-APA-LP) with the APA Feasibility Test (JLDP-APA-feas), the heuristic-based reduction (G-FP-APAh) and the APA simulation (FP-APA-sim), for a fixed number of processors equal to 16.

the number of tasks improves feasibility. This can be attributed to the high percentage of small tasks that are generated for a fixed utilization cap.

Overall, the experiments showed that the LP-based analysis is an effective technique for analyzing APA schedulers. It subsumes the other two reduction-based approaches by providing good results with reasonable computational costs. We also recognize the limitations of the analysis in terms of schedulability, but since this problem is correlated to the pessimism under global JLFP scheduling and also to priority and affinity assignment, we leave these open questions to future work.

7 Conclusion

In this paper, we investigated the schedulability analysis of real-time tasksets 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. For the general case of JLDP scheduling, we showed that APA JLDP scheduling is equivalent to global and clustered JLDP scheduling. The primary contribution of this paper, however, is the schedulability analyses for APA scheduling. The proposed exhaustive-reduction-based analysis is simple, reuses the extensive body of results for global scheduling already available, but does not scale well beyond five processors. In this regard, the heuristic-based analysis reduces the computation time significantly and, based on our evaluation results, its accuracy is similar to, though not quite equal to the exhaustive-reduction-based analysis. In contrast, the proposed LP-based analysis for FP schedulers is as good as the exhaustive-reduction-based analysis and at the same time achieves low runtime complexity, *i.e.*, it easily scales for problem sizes of up to thirty-two processors.

In summary, this paper establishes that APAs are useful from a scheduling point of view and proposes novel schedulability analysis methods for tasksets with APAs.

We hope to stir further research into the design of improved analysis techniques for APA scheduling and stronger models with more flexible migration strategies. For example, APAs do not place any restrictions on *when* migrations can take place. Therefore, another obvious generalization of the studied problem would be to interpret each α_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 jointly optimal processor affinity and priority assignments, as already highlighted in Section 6.

References

Alfieri RA (1998) Apparatus and method for improved CPU affinity in a multiprocessor system. US Patent 5,745,778

Anderson JH, Bud V, Devi UC (2005) An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In: Proceedings of the 17th Euromicro Conference on Real-Time Systems, ECRTS'05, pp 199–208

Andersson B, Jonsson J (2000) Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In: Proceedings of the Work-in-Progress Session of the 21st IEEE Real-Time Systems Symposium, RTSS'00

Andersson B, Raravi G, Bletsas K (2010) Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. In: Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS'10, pp 239–248

Audsley NC, Burns A, Richardson MF, Wellings AJ (1991) Hard real-time scheduling: The deadline-monotonic approach. In: Proceedings of the 1991 IEEE Workshop on Real-Time Operating Systems and Software, pp 133–137

Audsley NC, Burns A, Richardson MF, Tindell K, Wellings AJ (1993) Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal 8(5):284–292

Bado B, George L, Courbin P, Goossens J (2012) A semi-partitioned approach for parallel real-time scheduling. In: Proceedings of the 20th International Conference on Real-Time and Network Systems, RTNS'12, pp 151–160

Baker TP, Baruah SK (2007) Schedulability analysis of multiprocessor sporadic task systems. In: Handbook of Realtime and Embedded Systems, CRC Press

Baruah SK (2004) Partitioning real-time tasks among heterogeneous multiprocessors. In: Proceddings of the International Conference on Parallel Processing, ICPP'04, pp 467–474

Baruah SK (2007) Techniques for multiprocessor global schedulability analysis. In: Proceedings of the 28th IEEE Real-Time Systems Symposium, RTSS'07, pp 119–128

Baruah SK, Bini E (2008) Partitioned scheduling of sporadic task systems: an ILP-based approach. In: Conference on Design and Architectures for Signal and Image Processing, Bruxelles, Belgium, DASIP'08

Baruah SK, Brandenburg BB (2013) Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities. In: Proceedings of the 34th IEEE Real-Time Systems Symposium, RTSS'13, pp 160–169

Baruah SK, Cohen NK, Plaxton CG, Varvel DA (1996) Proportionate progress: A notion of fairness in resource allocation. Algorithmica 15:600–625

Bastoni A, Brandenburg BB, Anderson JH (2011) Is semi-partitioned scheduling practical? In: Proceedings of the 23rd Euromicro Conference on Real-Time Systems, ECRTS'11, pp 125–135

Bertogna M, Cirinei M (2007) Response-time analysis for globally scheduled symmetric multiprocessor platforms. In: Proceedings of the 28th IEEE Real-Time Systems Symposium, RTSS'07, pp 149–160

Brandenburg BB (2011) Scheduling and locking in multiprocessor real-time operating systems. PhD thesis, University of North Carolina

Burns A, Davis RI, Wang P, Zhang F (2012) Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme. Real-Time Systems 48:3–33

Calandrino JM, Anderson JH, Baumberger DP (2007) A hybrid real-time scheduling approach for large-scale multicore platforms. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems, ECRTS'07, pp 247 –258

Davis RI, Burns A (2011a) Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. Real-Time Systems 47(1):1–40

Davis RI, Burns A (2011b) A survey of hard real-time scheduling for multiprocessor systems. ACM Computing Surveys 43(4):35:1–35:44

Dertouzos ML, Mok AK (1989) Multiprocessor online scheduling of hard-real-time tasks. IEEE Transactions on Software Engineering 15(12):1497 –1506

Dorin F, Yomsi PM, Goossens J, Richard P (2010) Semi-partitioned hard real-time scheduling with restricted migrations upon identical multiprocessor platforms. CoRR abs/1006.2637

Easwaran A, Shin I, Lee I (2009) Optimal virtual cluster-based multiprocessor scheduling. Real-Time Systems 43(1):25–59

Eisenbrand F, Rothvoß T (2008) Static-priority real-time scheduling: Response time computation is NP-hard. In: Proceedings of the 29th IEEE Real-Time Systems Symposium, RTSS'08, pp 397–406

Eisenbrand F, Rothvoß T (2010) EDF-schedulability of synchronous periodic task systems is coNP-hard. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, pp 1029–1034

Emberson P, Stafford R, Davis RI (2010) Techniques for the synthesis of multiprocessor tasksets. In: Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, WATERS'10, pp 6–11

Fisher N, Goossens J, Baruah SK (2010) Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. Real-Time Systems 45(1-2):26–71

Foong A, Fung J, Newell D (2004) An in-depth analysis of the impact of processor affinity on network performance. In: Proceedings of the 12th IEEE International Conference on Networks, ICON'04, pp 244–250

Foong A, Fung J, Newell D, Abraham S, Irelan P, Lopez-Estrada A (2005) Architectural characterization of processor affinity in network processing. In: Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software, pp 207–218

Funk SH (2004) EDF scheduling on heterogeneous multiprocessors. PhD thesis, The University of North Carolina at Chapel Hill

Gálvez JJ, Ruiz PM, Skarmeta AFG (2010) Heuristics for scheduling on restricted identical machines. Tech. rep., University of Murcia, Spain

Guan N, Stigge M, Yi W, Yu G (2009) New response time bounds for fixed priority multiprocessor scheduling. In: Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS'09, pp 387–397

Gujarati A, Cerqueira F, Brandenburg BB (2013) Schedulability analysis of the linux push and pull scheduler with arbitrary processor affinities. In: Proceedings of the 25th Euromicro Conference on Real-Time Systems, ECRTS'13, pp 69–79

Harbour M, Palencia JC (2003) Response time analysis for tasks scheduled under EDF within fixed priorities. In: Proceedings of the 24th IEEE Real-Time Systems Symposium, RTSS'03, pp 200–209

Jang HC, Jin HW (2009) MiAMI: Multi-core aware processor affinity for TCP/IP over multiple network interfaces. In: Proceedings of the 17th IEEE Symposium on High Performance Interconnects, HOTI'13, pp 73–82

Joseph M, Pandya P (1986) Finding response times in a real-time system. The Computer Journal 29(5):390–395

Kato S, Yamasaki N, Ishikawa Y (2009) Semi-partitioned scheduling of sporadic task systems on multiprocessors. In: Proceedings of the 21st Euromicro Conference on Real-Time Systems, ECRTS'09, pp 249 –258

Lelli J, Lipari G, Faggioli D, Cucinotta T (2011) An efficient and scalable implementation of global EDF in Linux. In: Proceedings of the 7th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT'11, pp 6–15

Leung JYT, Li CL (2008) Scheduling with processing set restrictions: A survey. International Journal of Production Economics 116(2):251 – 262

Leung JYT, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance evaluation 2(4):237–250

Lisper B, Mellgren P (2001) Response-time calculation and priority assignment with integer programming methods. In: Proceedings of the Work-in-Progress and Industrial Sessions of the 13th Euromicro Conference on Real-Time Systems, ECRTS'01

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20(1):46–61

Lundberg L (1998) Multiprocessor scheduling of age constraint processes. In: Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications, RTCSA'98, pp 42–47

Markatos E, LeBlanc T (1992) Using processor affinity in loop scheduling on shared-memory multiprocessors. In: Proceedings of Supercomputing'92, pp 104–113

Mok AK (1983) Fundamental design problems of distributed systems for the hard-real-time environment. Tech. rep., Massachusetts Institute of Technology

Palencia J, Harbour MG (2005) Response time analysis of EDF distributed real-time systems. Journal of Embedded Computing 1(2):225–237

Reddy D, Koufaty D, Brett P, Hahn S (2011) Bridging functional heterogeneity in multicore architectures. SIGOPS Operating Systems Review 45(1):21–33

Salehi JD, Kurose JF, Towsley D (1995) Further results in affinity-based scheduling of parallel networking. University of Massachusetts, Amherst, MA

Zeng H, Di Natale M (2010) Improving real-time feasibility analysis for use in linear optimization methods. In: Proceedings of the 22nd Euromicro Conference on Real-Time Systems, ECRTS'10, pp 279–290