Accounting for Interrupts in Multiprocessor Real-Time Systems*

Björn B. Brandenburg, Hennadiy Leontyev, and James H. Anderson
The University of North Carolina at Chapel Hill

Abstract

The importance of accounting for interrupts in multiprocessor real-time schedulability analysis is discussed. Three interrupt accounting methods, two of which are newly described here, are analyzed and compared.

1 Introduction

System overheads such as time lost to task switches and scheduling decisions must be accounted for in real-time systems if temporal correctness is to be guaranteed [12, 24]. Of these overheads, *interrupts* are notoriously troublesome for real-time systems since they are not subject to scheduling and can significantly delay real-time tasks.

In work on uniprocessor real-time systems, methods have been developed to account for interrupts under the two most-commonly considered real-time scheduling policies: under *static-priority* scheduling, interrupts can be analyzed as higher-priority tasks [24], and under *earliest-deadline-first* (EDF) scheduling, schedulability can be tested by treating time lost to processing interrupts as a source of blocking [19].

Properly—but not too pessimistically—accounting for interrupts is even more crucial in multiprocessor real-time systems. Due to their increased processing capacity, such systems are likely to support much higher task counts, and since real-time tasks are usually invoked in response to interrupts, multiprocessor systems are likely to service interrupts much more frequently. Further, systematic pessimism in the analysis has a much larger impact on multiprocessors (see below).

Unfortunately, interrupts have not received sufficient attention in work on multiprocessor real-time systems. The first and, to the best of our knowledge, only published approach to date was proposed by Devi [13]. Devi presented a *quantum-centric* accounting method in which the length of the system's scheduling quantum is reduced to reflect time lost to overheads. In this paper, we consider this method, as well as two others, in the context of global scheduling algorithms. When the quantum-centric method is applied in this context, it is usually necessary to assume that all possible interrupts occur on all processors in every quantum. This assumption is obviously quite pessimistic and motivates the consideration of other approaches.

*Work supported by IBM, Intel, and Sun Corps.; NSF grants CNS 0834270, CNS 0834132, and CNS 0615197; and ARO grant W911NF-06-1-0425.

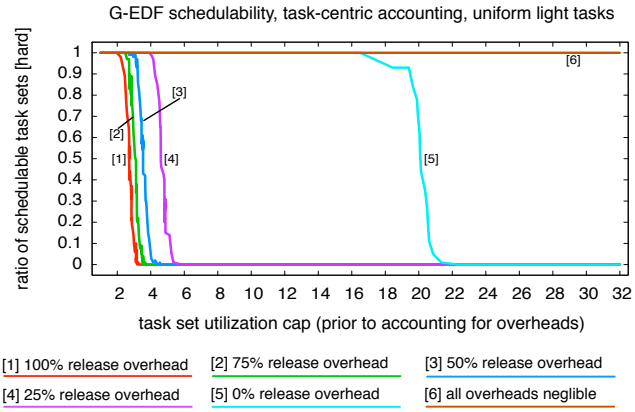

Figure 1: Hard real-time schedulability under G-EDF on a 32-processor platform assuming reduced release overhead, which is accounted for using the task-centric method. Note that all task sets are schedulable if all overheads are assumed to be negligible. (This graph corresponds to Figure 2(a) in [9]; see Sec. 5.)

Motivating example. In a recent case study on a 32-processor platform involving up to 600 light[1] tasks [9], the release overhead (*i.e.*, the time taken to process a timer interrupt and invoke a real-time task) of a *global EDF* (G-EDF) implementation was measured to exceed 50μs in the worst case. Given the system's quantum size of 1000μs, the quantum-centric method would have deemed any task set of 20 or more tasks unschedulable—with fewer tasks than processors, this is clearly excessively pessimistic.

In the above case study, a new, less pessimistic "task-centric" accounting method (see Sec. 4) was developed. However, even with "task-centric" accounting, G-EDF performed worse than expected. Suspecting high release overhead to be the cause, we conducted simulations to estimate performance assuming reduced overhead. Surprisingly, we found that even with a 75% reduction in release overhead, schedulability increases very little (see Fig. 1). However, the experiments also confirmed release overhead as the leading cause of reduced schedulability—performance improved dramatically assuming overhead-free releases. This discrepancy stems from quadratically-growing pessimism in the "task-centric method" (this phenomenon and a "processor-centric" accounting method that eliminates it are discussed in detail in Sec. 4).

This example shows that accurate accounting for overheads is crucial to multiprocessor real-time performance. Further,

[1]The utilization of the tasks was distributed uniformly in $[0.001, 0.1]$. Please see [9] for a detailed description of these experiments.

systematic pessimism has a large impact on multiprocessor systems due to high task counts.

Contributions. The contributions of this paper are as follows: **(i)** We highlight the importance of accurate interrupt accounting for multiprocessor real-time systems; **(ii)** we discuss a range of commonly-encountered interrupt sources in current multiprocessor systems; **(iii)** we show how interrupt accounting is fundamentally different from previous work on reduced-capacity scheduling; and **(iv)** we propose two new interrupt accounting methods ("task-centric" and "processor-centric") for both hard and soft real-time systems and briefly evaluate their effectiveness. Note that even though the "task-centric" method has been used in previous studies [8, 9], this paper is the first to present it in detail and argue its correctness.

The rest of the paper is organized as follows. Sec. 2 provides a detailed discussion of interrupts on multiprocessor systems, Sec. 3 formalizes the system model, Sec. 4 presents three approaches to accounting for interrupts, Sec. 5 provides a brief comparison of the quantum-centric and the proposed accounting methods, and Sec. 6 concludes.

2 Interrupts

To motivate our system model, we begin by providing a high-level overview of interrupts in a modern multiprocessor architecture. We focus on Intel's x86 architecture because it is in widespread use and well-documented [16, 17], but the discussion similarly applies to other multiprocessor architectures as well [25, 27].

Interrupts notify processors of asynchronous events and may occur between (almost) any two instructions. If an interrupt is detected, the processor temporarily pauses the execution of the currently-scheduled task and executes a designated *interrupt service routine* (ISR) instead. Obviously, this can lead to undesirable delays of the interrupted task; this must be accounted for in real-time schedulability analysis.

Most interrupts are *maskable*, *i.e.*, the processor can be instructed by the OS to delay the invocation of ISRs until interrupts are unmasked again. However, *non-maskable interrupts* (NMIs), which can be used for "watch dog" functionality to detect system hangs, cannot be suppressed by the OS [17].

In multiprocessor systems, some interrupts may be local to a specific processor (*e.g.*, register-based timers [27]), whereas others may be serviced by multiple or all processors.

Interrupts differ from normal preemptions in that a task cannot migrate while it is being delayed by an ISR, *i.e.*, a task cannot resume execution on another processor to reduce its delay. This limitation arises due to the way context switching is commonly implemented in OSs. For example, in Linux (and Linux-derived systems such as the one considered in [9]), there is only a single function in which context switching can be performed, and it is only invoked at the *end* of an ISR (if a preemption is required). From a software engineering point of view, limiting context switches in this way is desirable because it significantly reduces code complexity. In terms of performance, ISRs tend to be so short (in well-designed systems) that context-switching and migration costs dominate ISR execution times. Hence, delaying tasks is usually preferable to allowing migrations unless either migration and scheduling costs are negligible or ISR execution times are excessive.

Delays due to ISRs are fundamentally different from scheduling and preemption overheads: the occurrence of scheduling and preemption overheads are controlled by the OS and can be carefully scheduled not to occur at inopportune times. In contrast, ISRs execute with a statically-higher priority than any real-time task in the system and cannot be scheduled, *i.e.*, while interrupts can be temporarily masked by the OS, they cannot be selectively delayed[2] and are hence not subject to the scheduling policy of the OS.

Interrupt categories. Interrupts can be broadly categorized into four classes: *device interrupts* (DIs), *timer interrupts* (TIs), *cycle-stealing interrupts* (CSIs), and *inter-processor interrupts* (IPIs). We briefly discuss the purpose of each next.

DIs are triggered by hardware devices when a timely reaction by the OS is required or to avoid costly "polling" (see below).

TIs are used by the OS to initiate some action in the future. For example, TIs are used to support high-resolution delays ("sleeping") in Linux, and can be used for periodic job releases and to enforce execution time budgets.

CSIs are an artifact of how modern hardware architectures are commonly implemented and differ from the other categories in that they are neither controlled nor handled by the OS. CSIs are used to "steal" processing time for some component that is—from the point of view of the OS—hardware, but that is implemented as a combination of hardware and software (so called "firmware") and that lacks its own processor. CSIs are intended to be transparent from a logical correctness point of view, but of course do affect temporal correctness.[3] CSIs are usually non-maskable and the OS is generally unaware if and when CSIs occur. A well-known example for the use of CSIs is the system management mode (SMM) in Intel's x86 architecture [16, 17]: when a system management interrupt (SMI) occurs, the system switches into the SMM to execute ISRs stored in firmware. For example, on some chip sets the SMM is entered to control the speed of fans for cooling purposes. CSIs can also occur in architectures in which raw hardware access is mediated by a hypervisor (such as Sony's PlayStation 3 [20] and SUN's sun4v architecture [25])—the hypervisor may become active at any time to handle interrupts or perform services for devices "invisible" to the OS.

[2] Masking may create non-trivial timing dependencies because it usually affects physical *interrupt lines*, which are oftentimes shared among multiple interrupt sources.

[3] CSIs are especially problematic if the code that is being executed is unknown—for example, a CSI could flush instruction and data caches and thereby unexpectedly increase task execution costs.

In contrast to DIs, TIs, and CSIs, the final category considered, IPIs, are specific to multiprocessor systems. IPIs are used to synchronize state changes across processors and are generated by the OS. For example, the modification of memory mappings (*i.e.*, changes to address spaces) on one processor can require software-initiated TLB flushes on multiple processors [17, 18]. IPIs are also commonly used to cause a remote processor to reschedule.

Avoiding delays. There are three implementation choices that help to limit interrupt-related delays: Split interrupt handling, polling, and interrupt masking.

Split interrupt handling can be used to reduce the length of ISRs. With split interrupt handling, the work required to handle an interrupt is divided into two parts: a short ISR only acknowledges the interrupt and does the minimum amount of work necessary for correct operation, whereas the main work is carried out by an interrupt thread that is subject to OS scheduling [24]. However, even with split interrupt handling, some work, such as releasing jobs or activating interrupt threads, must be carried out in the ISRs themselves, and this work must be accounted for.

DIs can be avoided altogether through *polling*, whereby hardware devices are probed periodically for state changes and pending events. However, TIs are still required to invoke the scheduler and initiate polling periodically.

In embedded systems that execute real-time tasks in privileged mode, interrupts can be masked whenever a real-time job is executing. Once a job completes, interrupts are unmasked and pending interrupts handled. While this helps to make interrupts more predictable, it does not reduce the time lost to ISRs. Further, this approach does not apply to NMIs and, for security and safety reasons, it is highly undesirable to run real-time tasks in privileged mode. Additionally, if timing constraints are stringent, then increased latencies may be prohibitive. If IPIs are masked for prolonged times, then concurrency may be reduced while processors wait to synchronize.

Both polling and masking interrupts for prolonged times can drastically reduce the maximum I/O throughput by increasing device idle time. Such throughput losses may be economically undesirable or even unacceptable (especially in soft real-time systems). Finally, neither approach is a viable choice in the increasingly-relevant class of general purpose operating systems with real-time properties (such as real-time variants of Linux). Further, if a hardware platform makes use of CSIs, then, by definition, they cannot be avoided by any OS design.

To summarize, interrupts can delay real-time tasks and cannot be avoided completely on most (if not all) modern multiprocessor architectures; hence, they must be accounted for in schedulability analysis.

Bounding interference. Bounding ISR activations may be difficult in practice. The number of distinct interrupts is often limited (to reduce hardware costs), and hence interrupts may be shared among devices. Further, many devices and interrupts are commonly multiplexed among many logical tasks. For example, a single timer (and its corresponding ISR) is likely to be shared among multiple real-time tasks and potentially even best-effort tasks.

As a result, it may be impractical to characterize a system's worst-case interrupt behavior by modeling the individual (hardware) interrupts. Instead, it may be more illuminating to consider logical "interrupt sources" that cause one or more ISRs to be invoked in some pattern, but do not necessarily correspond to any particular device. This approach is formalized in the next section.

3 System Model

We consider the problem of scheduling a set of n implicit-deadline sporadic tasks $\tau = \{T_1, \ldots, T_n\}$ on m processors; we let $T_i(e_i, p_i)$ denote a task where e_i is T_i's *worst-case per-job execution time* and p_i is its *period*. $T_{i,j}$ denotes the j^{th} job ($j \geq 1$) of T_i. $T_{i,j}$ is released at $r_{i,j} \geq 0$ and should complete by its absolute deadline $d_{i,j} = r_{i,j} + p_i$. If $j > 1$, then $r_{i,j} \geq r_{i,j-1} + p_i$. If $T_{i,j}$ completes at time t, then its *tardiness* is $\max(0, t - d_{i,j})$. A task's tardiness is the maximum of the tardiness of any of its jobs. Even if $T_{i,j}$ misses its deadline $r_{i,j+1}$ is not altered. However, tasks are sequential: $T_{i,j+1}$ cannot start execution until $T_{i,j}$ completes. T_i's *utilization* is $u_i = e_i/p_i$; τ's *total utilization* is $U(\tau) = \sum_{i=1}^{n} u_i$. We assume $U(\tau) \leq m$; otherwise, tardiness may grow unboundedly [13].

Scheduler. In this paper, we assume preemptive G-EDF scheduling (*i.e.*, jobs with smaller $d_{i,j}$ values have higher priority). In an *event-driven* system, the scheduler is invoked whenever a job is released (to check if a preemption is required) or completes (to select the next job, if any). In contrast, in a *quantum-driven* system (see [13, 24] for an overview), the scheduler is invoked only at integer multiples of a *scheduling quantum* Q. Hence, job releases and completions may be processed with a delay of up to Q time units[4] and all task parameters must be integer multiples of Q.

Interrupts. An *interrupt source* causes ISRs to be invoked. An interrupt source is *local* if all invoked ISRs are serviced on the same processor, and *global* otherwise. When an ISR is invoked on a processor, the job currently running on that processor is temporarily *stopped* and its completion is delayed. In contrast to a regular preemption, a stopped job cannot migrate to another processor while the interfering ISR executes.

We consider a system with r global interrupt sources $I_1, \ldots, I_r$. Further, on each processor h, where $1 \leq h \leq m$, there are r_h local interrupt sources $I_1^h, \ldots, I_{r_h}^h$. We assume that for each interrupt source I_x (either global or local) there is a monotonic function $\mathsf{dbf}(I_x, \Delta)$ that bounds the maximum service time required by all ISRs invoked by I_x over an in-

[4] This delay can be accounted for by shortening a task's period (and hence relative deadline) by Q time units. A choice of $Q = 1$ms is common [13].

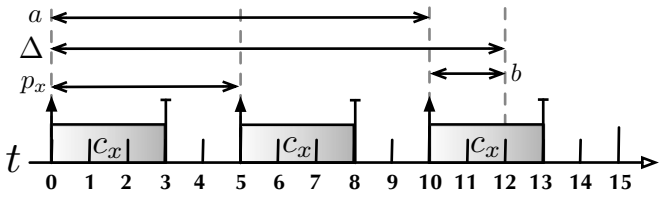

Figure 2: Illustration of (2) for $c_x = 3$, $p_x = 5$, and $\Delta = 12$, where $a = p_x \cdot \left\lfloor \frac{\Delta}{p_x} \right\rfloor$ and $b = \Delta - \left\lfloor \frac{\Delta}{p_x} \right\rfloor \cdot p_x$. At most $\left\lfloor \frac{\Delta}{p_x} \right\rfloor = 2$ complete ISR invocations execute in Δ (times 0–3 and 5–8), and $b = 2$ time units of a third invocation fit within Δ (time 10–12).

terval of length $\Delta \geq 0$. Additionally, we assume that, for each interrupt source (either local or global) I_x, $\mathsf{dbf}(I_x, \Delta)$ is upper-bounded by a linear function of Δ, *i.e.*, there exist $P(I_x) \leq 1$ and $R(I_x) \geq 0$ such that (1) below holds for each $\Delta \geq 0$.

$$\mathsf{dbf}(I_x, \Delta) \leq P(I_x) \cdot \Delta + R(I_x) \tag{1}$$

As an example, if a sporadic interrupt source I_x invokes an ISR of maximum length c_x at most once every p_x time units, then its demand bound function is given by

$$\mathsf{dbf}(I_x, \Delta) = \left\lfloor \frac{\Delta}{p_x} \right\rfloor \cdot c_x + \min\left(c_x,\ \Delta - \left\lfloor \frac{\Delta}{p_x} \right\rfloor \cdot p_x\right), \tag{2}$$

as illustrated in Fig. 2, in which case, $P(I_x) = c_x/p_x$ and $R(I_x) = c_x$.

We require that, in total, the time spent on interrupt handling does not exceed the capacity of one processor for sufficiently long time intervals. This assumption holds if

$$\sum_{k=1}^{r} P(I_k) + \sum_{h=1}^{m} \sum_{k=1}^{r_h} P(I_k^h) < 1. \tag{3}$$

Note that systems that violate (3) likely consist of sufficiently high processor and task counts to render global scheduling impractical. We expect such large systems to be scheduled using a *clustered* [9, 10] approach, wherein (3) applies only on a per-cluster basis (and hence is not a serious limitation).

Schedulability. In a *hard real-time system*, each job must complete before its absolute deadline and, in a *soft real-time system*, each job must have bounded maximum deadline tardiness. We are interested in developing a validation procedure—or *schedulability test*—for determining whether hard or soft real-time constraints are met for a task set τ that is scheduled on m processors using G-EDF in the presence of interrupts.

Many hard and soft real-time schedulability tests for G-EDF without interrupts have been proposed in prior work [1, 3, 6, 5, 14, 22]. In the next section, we describe three methods for incorporating interrupts in existing analysis, Devi's "quantum-centric" method and two new methods.

4 Schedulability Analysis

Because interrupt handlers effectively have higher priority than ordinary jobs, they may delay job completions. This can be accounted for in three different ways. Under *quantum-centric accounting* [13], interrupts are understood to reduce the "effective quantum length," *i.e.*, the service time available in each quantum to real-time jobs. Each task's worst-case execution cost is inflated to ensure completion given a lower-bound on the effective quantum length. Under *task-centric accounting*, interrupts are considered to extend each job's actual execution time and worst-case execution times are inflated accordingly. Under *processor-centric accounting*, task parameters remain unchanged and interrupts are considered to reduce the processing capacity available to tasks.

The first two methods are not G-EDF-specific—a task system is deemed schedulable if it passes an existing *sustainable* [4] schedulability test assuming inflated worst-case execution times for all tasks. Processor-centric accounting requires a two-step process: **(i)** reduce the processing capacity of the platform by accounting for the time consumed by interrupt handlers in the worst case; and **(ii)** analyze the schedulability of the original task set on the reduced-capacity platform. While general in principle, the reduced-capacity analysis used in Step (ii) has been developed only for a limited number of scheduling algorithms to date (G-EDF among them [21, 26]).

4.1 Quantum-Centric Accounting

Recall from Sec. 3 that under quantum-based scheduling all task parameters are multiples of a quantum size Q and that scheduling decisions are only made at multiples of Q. In practice, some processor time is lost due to system overheads during each quantum; the remaining time is called the *effective quantum length* Q'. With Devi's quantum-based accounting method [13], Q' is derived by assuming that all interrupt sources require maximum service time in each quantum, *i.e.*, on processor h

$$Q'_h = Q - \sum_{k=1}^{r_h} \mathsf{dbf}(I_k^h, Q) - \sum_{k=1}^{r} \mathsf{dbf}(I_k, Q). \tag{4}$$

Under G-EDF, it generally cannot be predicted on which processor(s) a job will execute, hence the system-wide effective quantum length is the minimum of the per-processor effective quantum lengths, *i.e.*, $Q' = \min\{Q'_h\}$. If $Q' > 0$, then task T_i's inflated worst-case execution time e'_i is given by

$$e'_i = Q \cdot \left\lceil \frac{e_i}{Q'} \right\rceil. \tag{5}$$

Obviously, τ is deemed unschedulable if $Q' \leq 0$ or $e'_i > p_i$ for any T_i. This technique does not depend on the scheduling algorithm in use—in fact, it has been used by Devi to analyze a range of algorithms [13]. A major limitation of the

quantum-centric method is that it tends to overestimate interrupt frequencies due to the short analysis interval Q.

4.2 Task-Centric Accounting

Based on the observation that $T_{i,j}$ is delayed by at most the total duration of ISRs that are invoked while $T_{i,j}$ executes, the task-centric method analyzes the complete interval during which a job $T_{i,j}$ can execute (instead of focusing on an individual quantum). In this paper, we describe this method in conjunction with G-EDF scheduling, but it can be applied similarly to other scheduling algorithms. Since the length of the analysis interval depends on $T_{i,j}$'s tardiness, the task-centric method is more involved in the soft real-time case. We start by first considering the hard real-time case.

The key concept behind the task-centric method is that, if jobs of $T_i(e_i, p_i)$ are delayed by at most δ_i, then T_i will meet all of its deadlines if $T_i'(e_i + \delta_i, p_i)$ meets all of its deadlines assuming no delays [15]. Hence, schedulability in the presence of ISR invocations can be checked with an existing schedulability test oblivious to interrupts if a bound on δ_i can be derived.

Definition 1. Let

$$C(\Delta) = \sum_{k=1}^{r} \mathsf{dbf}(I_k, \Delta) + \sum_{h=1}^{m} \sum_{k=1}^{r_h} \mathsf{dbf}(I_k^h, \Delta)$$

be a bound on the maximum time consumed by local and global ISRs during a time interval of length Δ.

Obviously, if $T_{i,j}$ completes by its deadline (*i.e.*, it is not tardy), then it can be *directly* delayed by ISRs for at most $C(d_{i,j} - r_{i,j}) = C(p_i)$ time units.[5] However, $T_{i,j}$ can also be *indirectly* delayed by ISRs that were invoked prior to $r_{i,j}$ by "pushing" processor demand of higher-priority jobs into $[r_{i,j}, d_{i,j})$. Both sources of delay can be accounted for by (pessimistically) assuming that *all* jobs are maximally delayed.

Theorem 1. A task system $\tau = \{T_1, \ldots, T_n\}$ is hard real-time schedulable with G-EDF in the presence of interrupts if $\tau' = \{T_1', \ldots, T_n'\}$, where $e_i' = e_i + C(p_i)$, passes a sustainable [4] hard real-time schedulability test.

Proof. Follows from the preceding discussion. □

Soft real-time schedulability. Our notion of soft real-time schedulability requires jobs to have bounded deadline tardiness. If all ISRs have zero cost, then the following holds.

Theorem 2. (Proved in [13].) If $U(\tau) \leq m$, then for each $T_i \in \tau$ there exists $b_i \geq 0$ such that T_i's maximum tardiness under G-EDF is at most b_i (b_i is given in [13]).

[5] Note that in upper-bounding δ_i by $C(p_i)$, local interrupts on *all* processors are considered. A tighter bound on δ_i can be derived if T_i is known to migrate at most $\eta < m$ times: only the $\eta + 1$ processors for which $\sum_{k=1}^{r_h} \mathsf{dbf}(I_k^h, \Delta)$ is maximized must be accounted for in this case.

In the absence of interrupts, every job $T_{i,j}$ is known to only execute within $[r_{i,j}, d_{i,j} + b_i)$. However, in contrast to the hard real-time case, the analysis interval changes in the presence of interrupts since inflating e_i also inflates b_i. Thus, an iterative approach such as the following procedure is required to break the cyclic dependency between the tardiness bound and delays due to ISRs in the soft real-time case.

initialize $b_i' := 0$ for all i
do
  set $b_i^o := b_i'$ for all i
  set $e_i' := e_i + C(p_i + b_i^o)$ for all i
  set $\tau' := \{T_1', \ldots, T_n'\}$ where $T_i' = (e_i', p_i)$
  compute b_i' for all i with Theorem 2 based on τ'
while ($b_i' \neq b_i^o$ **and** $e_i' < p_i$) for all i **and** $U(\tau') \leq m$

Theorem 3. If $U(\tau') \leq m$ and $e_i' \leq p_i$ for all $T_i \in \tau$ after the above procedure terminates, then deadline tardiness of each T_i is at most b_i' under G-EDF scheduling in the presence of interrupts.

Proof. Follows from the preceding discussion. □

Discussion. While often less pessimistic than the quantum-centric method, the task-centric method is also likely to overcharge for interrupts. In fact, the cause of the disappointing (lack of significant) performance improvements observed in Fig. 1 is utilization loss inherent in the task-centric method. This loss is quadratic in the number of tasks, as is shown next.

Definition 2. Let $c_r > 0$ denote the worst-case execution time of a job-releasing ISR, *i.e.*, the worst-case overhead incurred due to a single job release.

Theorem 4. Consider an event-triggered system τ of n sporadic tasks, where each job is released by an interrupt of cost c_r, where $\min\{p_i\} > c_r > 0$ and $\max\{p_i\}$ is bounded by some constant. Under task-centric interrupt accounting, $U(\tau') = U(\tau) + \Omega(n^2)$, where τ' denotes the inflated task set as defined in Theorems 1 and 3 (respectively).

Proof. First, we consider the hard real-time case. By Theorem 1 and Definition 1,

$$e_i' = e_i + C(p_i) = e_i + \sum_{k=1}^{r} \mathsf{dbf}(I_k, p_i) + \sum_{h=1}^{m} \sum_{k=1}^{r_h} \mathsf{dbf}(I_k^h, p_i).$$

Without loss of generality, assume that the only sources of interrupts $I_1, \ldots, I_n$ are sporadic job releases by $T_1, \ldots, T_n$, where I_i corresponds to releases of T_i. Hence $e_i' = e_i + \sum_{k=1}^{n} \mathsf{dbf}(I_k, p_i)$ and thus

$$U(\tau') = U(\tau) + \sum_{i=1}^{n} \frac{\sum_{k=1}^{n} \mathsf{dbf}(I_k, p_i)}{p_i}.$$

By (2) and Definition 2,

$$\mathsf{dbf}(I_k, p_i) = \left\lfloor \frac{p_i}{p_k} \right\rfloor \cdot c_r + \min\left(c_r,\ p_i - \left\lfloor \frac{p_i}{p_k} \right\rfloor \cdot c_r\right).$$

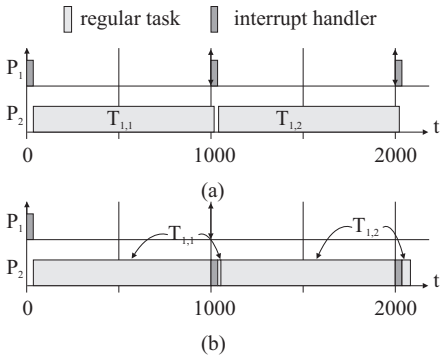

Figure 3: Interrupt handling scenarios from Example 1.

If $p_i < p_k$, then $\mathsf{dbf}(I_k, p_i) = \min(c_r,\ p_i) = c_r$; otherwise, $\mathsf{dbf}(I_k, p_i) \geq \left\lfloor \frac{p_i}{p_k} \right\rfloor \cdot c_r \geq c_r$. Hence, $\mathsf{dbf}(I_k, p_i) \geq c_r$, and therefore

$$U(\tau') \geq U(\tau) + \sum_{i=1}^{n} \frac{n \cdot c_r}{p_i} \geq U(\tau) + \frac{n^2 \cdot c_r}{\max\{p_i\}}.$$

This establishes a lower bound for the hard real-time case. The same lower bound applies in the soft real-time case as well since $C(p_i + b_i) \geq C(p_i)$ due to the monotonicity of $\mathsf{dbf}(I_k, \Delta)$ (recall that, by definition, $b_i \geq 0$). □

If each (of n) tasks releases a constant number of jobs over some interval, then the time lost to job releases over the interval is $O(n)$. Hence, Theorem 4 shows that the task-centric method asymptotically overestimates the time lost to interrupt processing. For sufficiently large n, reducing the release overhead c_r has comparably little impact—that is, optimizing the OS implementation does not help to mitigate asymptotic inefficiencies in the analysis. Next, we discuss the remaining processor-centric interrupt accounting method, which is designed to overcome this problem.

4.3 Processor-Centric Accounting

The final method, which involves subtracting the ISR time from the total available processing capacity, introduces several difficulties.

First, even though an ISR makes a processor unavailable to ordinary real-time jobs, the currently-running job cannot migrate to another processor. This differs from how limited processing capacity has been treated in prior work [21, 23, 26]. Second, if a job is released as a result of an ISR, then it cannot be scheduled on any processor before the respective ISR completes. Both aspects are illustrated by the following example, in which tardiness may grow unboundedly even though tardiness is deemed bounded using traditional reduced-capacity analysis [21].

Example 1. Consider the task set $\tau = \{T_1(999, 1000)\}$ scheduled on two processors. Suppose that the only ISR in the system, which releases T_1's jobs, is invoked every 1000 time units and executes for at most 2 time units. In the schedule shown in Fig. 3(a), job $T_{1,1}$, released at time 0, is not available for scheduling before time 2. At time 2, $T_{1,1}$ is scheduled on processor P_2 and completes at time 1001, so its tardiness is 1. At time 1000, job $T_{1,2}$ arrives and the ISR is invoked on processor P_1 so job $T_{1,2}$ becomes available to the scheduler at time 1002 and completes at time 2001. Also, running jobs of task T_1 are not paused in this schedule. In contrast to this, in the schedule in Fig. 3(b), the second interrupt is handled by processor P_2 and thus preempts job $T_{1,1}$, rendering both processors unavailable to T_1. If the ISR is invoked on the processor that schedules a job of T_1, then the tardiness will grow unboundedly because only 998 execution units are available to T_1 while it requests 999 time units every 1000 time units (assuming that future jobs are released periodically).

In order to re-use results for platforms with limited processor availability, we assume that, when an interrupt occurs, *all* processors become unavailable to jobs in τ for the duration of an ISR. While pessimistic, the above example illustrates that this global capacity reduction is required in order to achieve general applicability of the analysis. We next introduce some definitions to reason about platforms with limited processing capacity.

Definition 3. Let $\mathsf{supply}_h(t, \Delta)$ be the total amount of processor time available on processor h to the tasks in τ during the interval $[t, t + \Delta)$.

Definition 4. To deal with limited processor supply, the notion of *service functions* has been proposed [11]. The service function $\beta_h^l(\Delta)$ lower-bounds $\mathsf{supply}_h(t, \Delta)$ for each time $t \geq 0$. We require

$$\beta_h^l(\Delta) \geq \max(0, \widehat{u_h} \cdot (\Delta - \sigma_h)), \tag{6}$$

for $\widehat{u_h} \in (0, 1]$ and $\sigma_h \geq 0$.

In the above definition, the superscript l stands for "lower bound," $\widehat{u_h}$ is the total long-term utilization available to the tasks in τ on processor h, and σ_h is the maximum length of time when the processor can be unavailable. Note that, if processor h is fully available to the tasks in τ, then $\beta_h^l(\Delta) = \Delta$.

Some schedulability tests for limited processing capacity platforms require that the total time available on *all* processors be known [26].

Definition 5. Let $\mathsf{Supply}(t, \Delta) = \sum_{h=1}^{m} \mathsf{supply}_h(t, \Delta)$ be the cumulative processor supply during the interval $[t, t+\Delta)$. Let $\mathcal{B}(\Delta)$ be the guaranteed total time that all processors can provide to the tasks in τ during any time interval of length $\Delta \geq 0$.

If lower bounds on individual processor supply are known, then we can compute a lower bound on the total supply using the following trivial claim.

Claim 1. If individual processor service functions $\beta_h^l(\Delta)$ are known, then $\mathsf{Supply}(t, \Delta) \geq \mathcal{B}(\Delta)$, where $\mathcal{B}(\Delta) = \sum_{h=1}^{m} \beta_h^l(\Delta)$.

In the remainder of this section, we assume that individual processor service functions are known. We will derive expressions for them later in Sec. 4.4.

Hard real-time schedulability of τ on a platform with limited supply can be checked using results from [26]. This paper presents a sufficient pseudo-polynomial test that checks whether the time between any job's release time and its completion time does not exceed a pre-defined bound. This test involves calculating the minimum guaranteed supply for different values of Δ.

To test soft real-time schedulability of τ on a platform with limited supply we use the following definition and theorem.

Definition 6. Let $U_L(y)$ be the sum of $\min(|\tau|, y)$ largest task utilizations.

Theorem 5. (Proved in [21]) Tasks in τ have bounded deadline tardiness if the inequalities (7) and (8) below hold.

$$U(\tau) \leq \sum_{h=1}^{m} \widehat{u_h} \tag{7}$$

$$\sum_{h=1}^{m} \widehat{u_h} > \max(H-1, 0) \cdot \max(u_i) + U_L(m-1), \tag{8}$$

where H is the number of processors for which $\beta_h^l(\Delta) \neq \Delta$.

In the above theorem, if (7) does not hold, then the long-term execution requirement for tasks will exceed the total long-term guaranteed supply, and hence, the system will be overloaded. On the other hand, (8) implicitly restricts the maximum per-task utilization in τ due to the term $\max(H-1, 0) \cdot \max(u_i)$, which could be large if the maximum per-task utilization is large. This is especially the case if all processors can be unavailable to tasks in τ, *i.e.*, $H = m$. Since we assumed that all processors are unavailable to τ for the duration of an ISR, this may result in pessimistically claiming a system with large per-task utilizations to be unschedulable. In the next section, in Example 2, we show that such a penalty may be unavoidable.

4.4 Deriving β Service Functions

We now establish a lower bound on the supply provided by processor h to τ over an interval of length Δ.

Definition 7. Let $F = \sum_{k=1}^{r} P(I_k) + \sum_{h=1}^{m} \sum_{k=1}^{r_h} P(I_k^h)$ and $G = \sum_{k=1}^{r} R(I_k) + \sum_{h=1}^{m} \sum_{k=1}^{r_h} R(I_k^h)$.

Lemma 1. If interrupts are present in the system, then any processor h can supply at least

$$\beta_h^l(\Delta) = \max(0, \Delta - C(\Delta)) \tag{9}$$

time units to the tasks in τ over any interval of length Δ. Additionally, (6) holds for $\widehat{u_h} = 1 - F$ and $\sigma_h = \frac{G}{1-F}$.

Proof. We first prove (9). By Definition 3, processor h provides $\mathsf{supply}_h(t, \Delta)$ time units to τ during the interval $[t, t+\Delta)$. By our assumption, processor h is unavailable for the total duration of all local and global ISRs invoked during the interval $[t, t+\Delta)$. From this and Definition 1, we have

$$\mathsf{supply}_h(t, \Delta) \geq \Delta - C(\Delta)$$

Because $\mathsf{supply}_h(t, \Delta)$ cannot be negative, (9) follows from Definition 4. Our remaining proof obligation is to find $\widehat{u_h}$ and σ_h such that (6) holds. From (9), we have

$$\begin{aligned}
\beta_h^l(\Delta) &= \max(0, \Delta - C(\Delta)) \\
&\quad \{\text{by Definition 1}\} \\
&\geq \max\left(0, \Delta - \left[\sum_{k=1}^{r} \mathsf{dbf}(I_k) + \sum_{h=1}^{m}\sum_{k=1}^{r_h} \mathsf{dbf}(I_k^h)\right]\right) \\
&\quad \{\text{by (1)}\} \\
&\geq \max\left(0, \Delta - \left[\sum_{k=1}^{r} (P(I_k)\cdot\Delta + R(I_k)) \right.\right. \\
&\quad \left.\left. + \sum_{h=1}^{m}\sum_{k=1}^{r_h} (P(I_k^h)\cdot\Delta + R(I_k^h))\right]\right) \\
&\quad \{\text{by Definition 7}\} \\
&= \max(0, \Delta - (F\cdot\Delta + G)) \\
&\quad \{\text{by the definition of } \widehat{u_h} \text{ and } \sigma_h \\
&\quad \text{in the statement of the lemma}\} \\
&\geq \max(0, \widehat{u_h} \cdot (\Delta - \sigma_h)).
\end{aligned}$$

By Definition 7 and (3), $\widehat{u_h}$ as defined in the statement of the lemma is positive. □

We next illustrate soft real-time schedulability analysis of a system with interrupts using Lemma 1 and Theorem 5.

Example 2. Consider the system from Example 1. The maximum tardiness for T_1's jobs may or may not be bounded depending on how interrupts are dispatched. We now analyze this task system using Theorem 5. By (2), the only global interrupt source I_1 considered has $\mathsf{dbf}(I_1, \Delta) = \left\lfloor \frac{\Delta}{1000} \right\rfloor \cdot 2 + \min\left(2,\ \Delta - \left\lfloor \frac{\Delta}{1000} \right\rfloor \cdot 2\right)$. The parameters $P(I_1)$ and $R(I_1)$ for which (1) holds are 0.002 and 2, respectively. Setting these parameters into Lemma 1, we find $\widehat{u_h} = 0.998$, and $\sigma_h = 2.004$ for $h = 1$ and 2. Applying Theorem 5 to this configuration, we find that (8) does not hold because $\sum_{h=1}^{m} \widehat{u_h} = 2 \cdot 0.998 < \max(2-1, 0) \cdot u_1 + u_1 = 2 \cdot 0.999$. Thus, bounded tardiness is not guaranteed for τ.

5 Experimental Evaluation

To assess the effectiveness of the three interrupt accounting approaches described above, we conducted an experimental evaluation in which the methods were compared based on how

many randomly generated task sets could be claimed schedulable (both hard and soft) if interrupts are accounted for using each method.

Experimental setup. Similarly to the experiments previously performed in [9], we used distributions proposed by Baker [2] to generate task sets randomly. Task periods were uniformly distributed over [10ms, 100ms]. Task utilizations were distributed differently for each experiment using three uniform and three bimodal distributions. The ranges for the uniform distributions were [0.001, 0.1] (*light*), [0.1, 0.4] (*medium*), and [0.5, 0.9] (*heavy*). In the three bimodal distributions, utilizations were distributed uniformly over either [0.001, 0.5) or [0.5, 0.9] with respective probabilities of 8/9 and 1/9 (*light*), 6/9 and 3/9 (*medium*), and 4/9 and 5/9 (*heavy*).

We considered ISRs that were previously measured on a 32-processor platform in [9]; namely, the job release ISR (global) and the per-processor tick ISR signaling the beginning of a new quantum (local; $Q = 1000\mu$s). We assumed worst-case (average-case) ISR costs (from [9]) as given in Table 1 when testing hard (soft) schedulability. As we are only concerned with interrupt accounting, all other sources of overhead (scheduling, cache-related, *etc.*) were considered negligible.[6]

To assess the impact (or lack thereof) of possible improvements in the OS implementation, we tested each method twice: once assuming full ISR costs as given in Table 1, and once assuming ISR costs had been reduced by 80%.

For each task set, we applied each of the accounting methods as described in Sec. 4. To determine whether an inflated task system τ' is hard real-time schedulable under the quantum-centric and task-centric methods, we used all major published sufficient (but not necessary) hard real-time schedulability tests for G-EDF [1, 3, 5, 6, 14] and deemed τ' to be schedulable if it passes at least one of these five tests. To determine hard real-time schedulability under the processor-centric method, we used Shin *et al.*'s *Multiprocessor Resource* (MPR) test [26] and chose $\max\{p_i\}$ as the MPR period for analysis purposes. As an optimization, we used the supply-bound function given in (10) below, which is less pessimistic than the (more general) one given in [26].

$$\mathrm{sbf}(\Delta) = m\left(\Delta - \sum_{h=1}^{m}\sum_{k=1}^{r_h}\mathsf{dbf}(I_k^h,\Delta) - \sum_{k=1}^{r}\mathsf{dbf}(I_k,\Delta)\right) \tag{10}$$

Note that in [26] the supply-bound function is derived based on a given period and supply. In contrast, we compute the supply for a chosen period with (10) (for $\Delta = \max\{p_i\}$).

Our experimental results (discussed next) are shown in Figs. 4 and 5. Sampling points were chosen such that the sampling density is high in areas where curves change rapidly. For each sampling point, we tested 1,000 task sets, for a total of 1,736,000 task sets.

[6] An implementation study considering full overheads including interrupts (and other implementation choices beyond the scope of this paper) has recently been completed [7].

ISR	Worst-Case	Average-Case
tick	$3.043 + 0.003 \cdot n$	$2.080 + 0.002 \cdot n$
job release	$45.025 + 0.314 \cdot n$	$5.840 + 0.127 \cdot n$

Table 1: Worst-case and average-case ISR invocation costs under G-EDF (in μs) as a function of the task set size n (from [9]).

Results (100%). Fig. 4 shows the ratio of task sets claimed hard real-time schedulable by the three methods. In all scenarios, the quantum-centric method shows consistently disappointing performance. In Fig. 4(a), which corresponds to the motivating example in Sec. 1, even with utilization less than two, most task sets were deemed unschedulable using this method. The processor-centric method works best if there are no heavy tasks (see Fig. 4(a)). The task-centric method under these conditions is pessimistic due to large inflation costs in accordance with Theorem 4. As the number of tasks decreases (the average per-task utilization increases), the performance of the task-centric method improves significantly—in fact, it is the best choice for all but one of the tested distributions (see insets (b)–(f) of Fig. 4).

Fig. 5 shows soft real-time schedulability results. For the uniform light and medium utilization ranges, the performance of the processor-centric method is superior (see insets (a) and (c) of Fig. 5). This is because (7) and (8) in Theorem 5 are likely to hold in the absence of heavy tasks. In contrast, the processor-centric method is exceptionally pessimistic in the case of the uniform heavy distribution (see Fig. 5(e)). This is because we assumed that all processors are not available for the duration of an ISR, which leads to a violation of (8) if heavy tasks are present, due to the term $\max(H-1,0)\cdot\max(u_i)$ (note that $H = m$). For the same reason, task sets are pessimistically claimed unschedulable under the bimodal utilization distributions ((b), (d), and (f) of Fig. 5). In fact, the processor-centric method offers no advantage over the quantum-centric method in the soft real-time case if heavy tasks are present at all. This is in stark contrast to the task-centric method, which consistently performs best unless task counts are high.

Results (20%). The experiments assuming reduced release overhead reveal four major trends: **(i)** reducing overhead helps only little to overcome the task-centric method's asymptotic growth (Fig. 4(a) and Fig. 5(a)); **(ii)** the processor-centric method performs equally badly assuming either 100% or 20% overhead if heavy tasks are present (see insets (b) and (d)–(f) of both Fig. 4 and Fig. 5); **(iii)** even with reduced overhead, the quantum-centric method is not competitive in the hard real-time case; and **(iv)** the quantum-centric method is very competitive in the soft real-time case if overhead is reduced.

The results show that the choice of interrupt accounting method can have significant impact on schedulability. Further, it appears worthwhile to improve both OS implementations

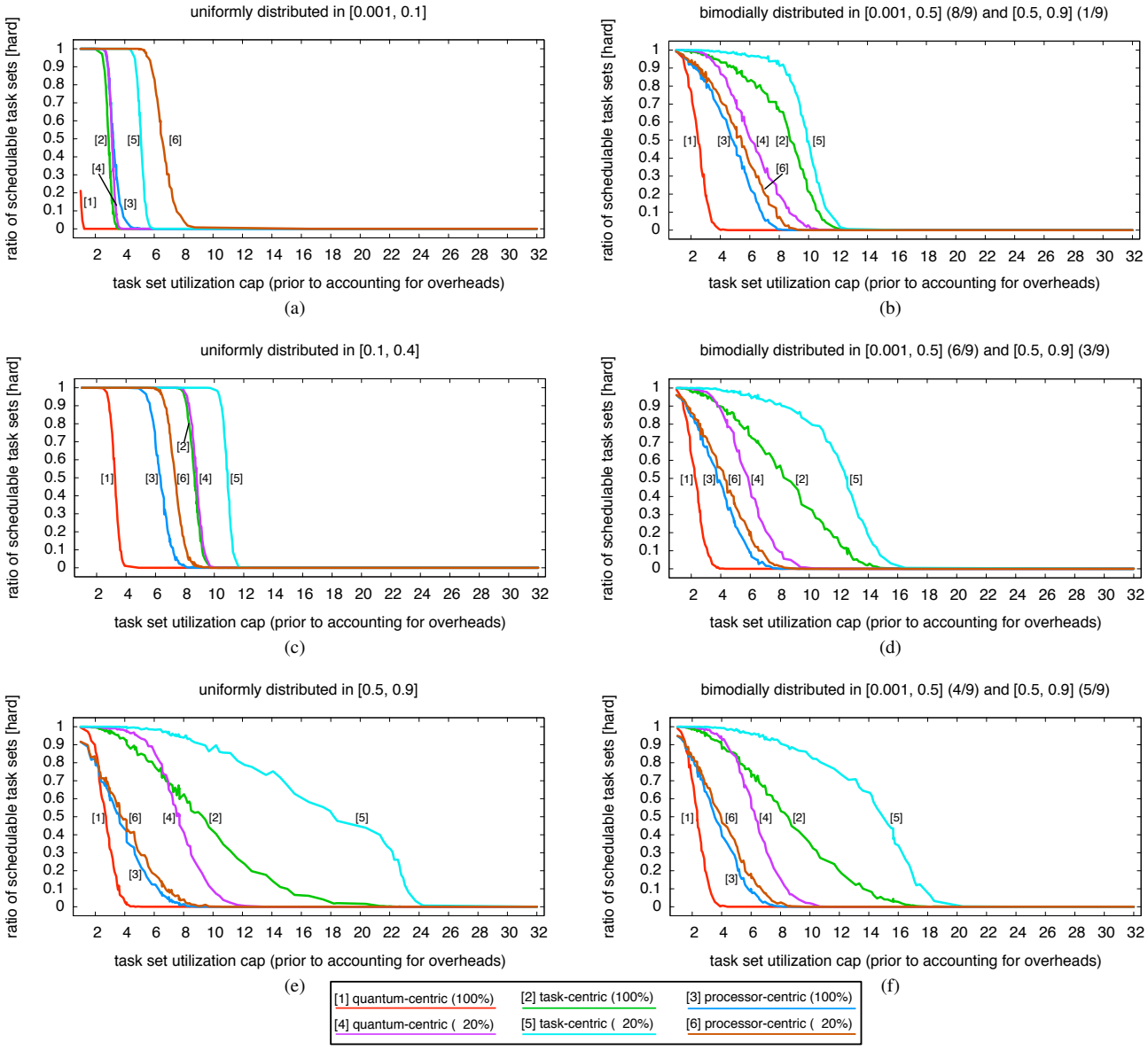

Figure 4: Hard schedulability results for **(a)**–**(b)** light, **(c)**–**(d)** medium, and **(e)**–**(f)** heavy utilization distributions. Results assuming uniform distributions are depicted in the left column (insets **(a)**, **(c)**, and **(e)**); results assuming bimodal utilization distributions are depicted in the right column (insets **(b)**, **(d)**, and **(f)**).

and analysis techniques to reduce inherent pessimism.

6 Conclusion

This paper discussed various interrupt sources in multiprocessor real-time systems. Two new approaches to accounting for interrupt-related delays under G-EDF scheduling for both hard and soft real-time systems were presented: the task-centric method and the processor-centric method. In an empirical comparison, the task-centric method performed well in most of the tested scenarios; however, it is subject to utilization loss that is quadratic in the number of tasks. Hence, it has inferior performance for large task sets. In contrast, the processor-centric method performed well for task systems with many light tasks, but yielded overly pessimistic results in the presence of heavy tasks. In all scenarios assuming 100% overhead, at least one of the two new methods performed significantly better than the previously-proposed quantum-centric method.

In future work, we would like to explore how the OS implementation can be adjusted to better work with existing multiprocessor real-time analysis—if accounting for delays due to interrupts is problematic, then how can we prevent interrupts from delaying real-time tasks? Further, we would like to refine the processor-centric method to be less pessimistic with regard

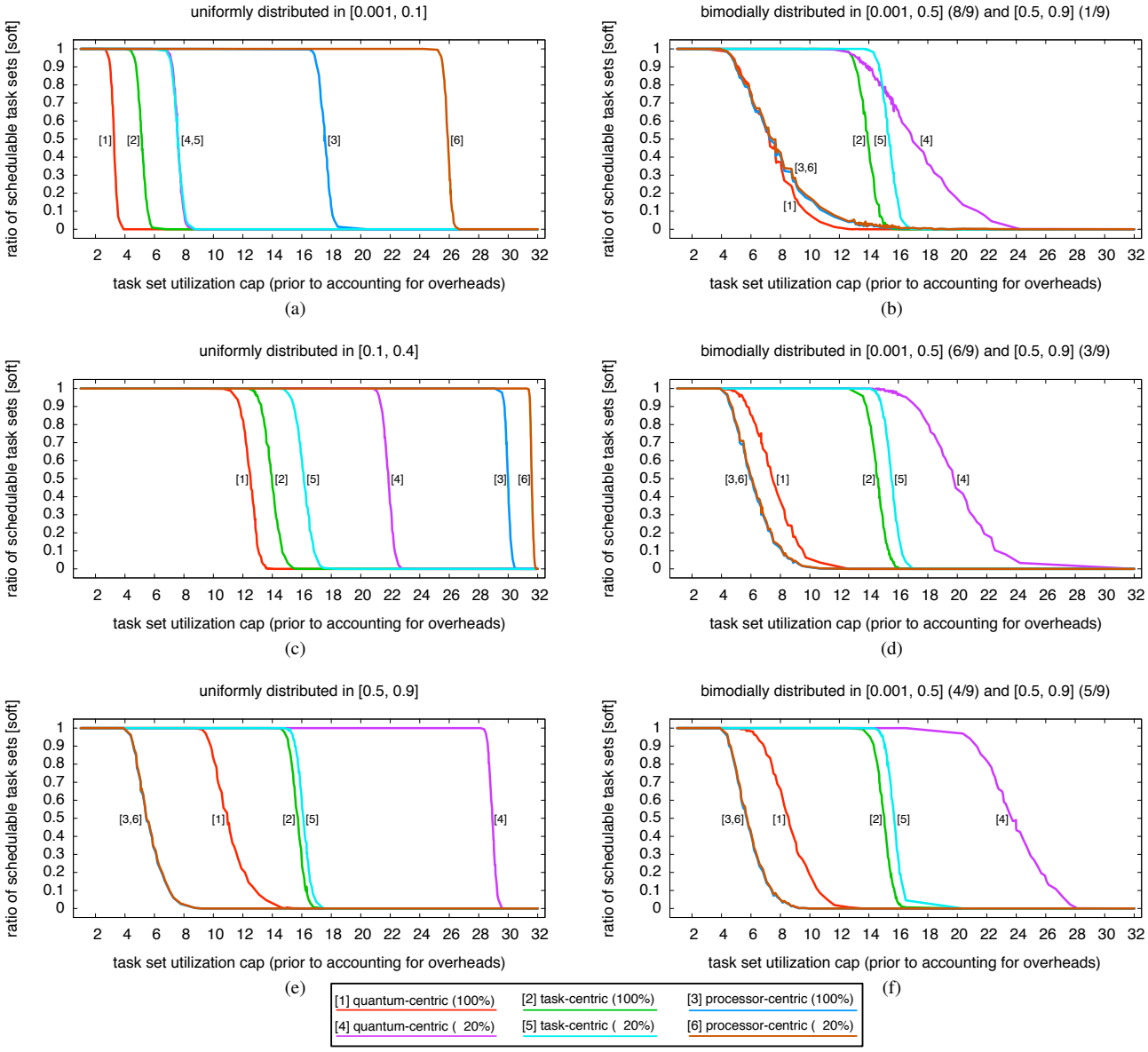

Figure 5: Soft schedulability results for **(a)**–**(b)** light, **(c)**–**(d)** medium, and **(e)**–**(f)** heavy utilization distributions. Results assuming uniform distributions are depicted in the left column (insets **(a)**, **(c)**, and **(e)**); results assuming bimodal utilization distributions are depicted in the right column (insets **(b)**, **(d)**, and **(f)**).

to both maximum task utilization and reductions in supply.

References

[1] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 120–129, 2003.

[2] T. Baker. A comparison of global and partitioned EDF schedulability tests for multiprocessors. Technical Report TR-051101, Florida State University, 2005.

[3] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*, pages 119–128, 2007.

[4] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 159–168, 2006.

[5] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 209–218, 2005.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, 2009.

[7] B. Brandenburg and J. Anderson. On the implementation of global real-time schedulers. Manuscript.

[8] B. Brandenburg and J. Anderson. A comparison of the M-

PCP, D-PCP, and FMLP on LITMUS$^{\mathrm{RT}}$. In *Proceedings of the 12th International Conference On Principles Of Distributed Systems*, pages 105–124, 2008.

[9] B. Brandenburg, J. Calandrino, and J. Anderson. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169, 2008.

[10] J. Calandrino, J. Anderson, and D. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 247–256, 2007.

[11] S. Chakraborty, Y. Liu, N. Stoimenov, L. Thiele, and E. Wandeler. Interface-based rate analysis of embedded systems. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 25–34, 2006.

[12] D. Cofer and M. Rangarajan. Formal verification of overhead accounting in an avionics RTOS. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, pages 181–190, 2002.

[13] U. Devi. *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, University of North Carolina, Chapel Hill, North Carolina, 2006.

[14] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, 2003.

[15] R. Ha and J. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pages 162–171, 1994.

[16] Intel Corp. *Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 1: Basic Architecture*. Intel Corp., 2008.

[17] Intel Corp. *Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 3: System Programming Guide*. Intel Corp., 2008.

[18] Intel Corp. *TLBs, Paging-Structure Caches, and Their Invalidation*. Intel Corp., 2008.

[19] K. Jeffay and D. Stone. Accounting for interrupt handling costs in dynamic priority task systems. *Proceedings of the 14th Real-Time Systems Symposium*, pages 212–221, 1993.

[20] J. Kurzak, A. Buttari, P. Luszczek, and J. Dongarra. The playstation 3 for high-performance scientific computing. *Computing in Science & Engineering*, 10(3):84–87, 2008.

[21] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*, pages 413–422, 2007.

[22] H. Leontyev and J. Anderson. A unified hard/soft real-time schedulability test for global EDF multiprocessor scheduling. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 375–384, 2008.

[23] H. Leontyev, S. Chakraborty, and J. Anderson. Multiprocessor extensions to real-time calculus. Manuscript, 2009.

[24] J. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[25] Ashley Saulsbury. *UltraSPARC Virtual Machine Specification*. SUN Corp., 2008.

[26] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems*, pages 181–190, 2008.

[27] D. Weaver and T. Germond, editors. *The SPARC Architecture Manual. Version 9*. PTR Prentice Hall, 1994.

A An Optimization for Timer Ticks

In this appendix, we present an improved accounting method for *periodic*, *replicated* interrupts (*e.g.*, timer ticks) to be used in conjunction with the task-centric method. An interrupt is periodic if its minimum invocation separation equals its maximum invocation separation, and replicated if a corresponding ISR is invoked on *each* processor. Note that we do not assume that ISRs are invoked on each processor simultaneously (*i.e.*, "staggered" ISR invocations are allowed), but invocations on each processor must be periodic (*i.e.*, the "staggering" is constant).

Recall from Sec. 4.2 that under task-centric accounting, the execution cost of a task $T_i = (e_i, p_i)$ is inflated to account for all local ISR invocations on all processors that may occur during the maximum interval in which a single job of T_i can be active, *i.e.*, a job of T_i is charged for all delays in the interval in which it *might* execute. In the general case, this (likely very pessimistic) charge is unavoidable because a job may execute on all processors (due to migrations) and be delayed by all invoked ISRs on each processor (due to adverse timing of ISR invocations). In fact, such a scenario is trivial to construct assuming sporadic interrupt sources (*i.e.*, no *maximum* separation of ISR invocations) and sporadic job releases.

Periodic ISR delays. Periodic interrupts differ from the more general model discussed in Sec. 3 in that we can assume an exact separation of ISR invocations, which enables us to apply techniques of classic response-time analysis (see [24] for an introduction). Let I_x denote a periodic, replicated interrupt source that, on each processor, triggers every p_x time units an ISR that executes for at most c_x time units. Initially assume that a job $T_{i,j}$ is not preempted while it executes, and assume that e_i already accounts for all other sources of delays (*e.g.*, global and local interrupts). Then, based on the response-time analysis of an equivalent two-task system under static-priority scheduling, $T_{i,j}$'s inflated execution cost is given by the smallest e'_i that satisfies

$$e'_i = e_i + \left\lceil \frac{e'_i}{p_x} \right\rceil \cdot c_x. \tag{11}$$

Migrations. If $T_{i,j}$ migrates, then (11) does not hold anymore—each time $T_{i,j}$ migrates, it might incur one additional ISR invocation due to ISR invocations not occurring simultaneously on all processors. Similarly, if $T_{i,j}$ is preempted, then it might also incur one additional ISR invocation when it resumes because its completion was delayed. Hence, if $T_{i,j}$ is preempted or migrates at most η times, then $T_{i,j}$'s inflated execution cost is given by the smallest e'_i that satisfies

$$e'_i = e_i + \left(\left\lceil \frac{e'_i}{p_x} \right\rceil + \eta \right) \cdot c_x \tag{12}$$

(formal proof—via induction over the number of intervals of consecutive execution—omitted due to space constraints).