A Hierarchical Multiprocessor Bandwidth Reservation Scheme with Timing Guarantees *

Hennadiy Leontyev
James H. Anderson
Department of Computer Science, The University of North Carolina at Chapel Hill

Abstract

A multiprocessor scheduling scheme is presented for supporting hierarchical containers that encapsulate sporadic soft and hard real-time tasks. In this scheme, each container is allocated a specified bandwidth, which it uses to schedule its children (some of which may also be containers). This scheme is novel in that, with only soft real-time tasks, no utilization loss is incurred when provisioning containers, even in arbitrarily deep hierarchies. Presented experiments show that the proposed scheme performs well compared to conventional real-time scheduling techniques that do not provide container isolation.

1 Introduction

In the Linux community, two recent developments have occurred that are of relevance to real-time software design processes. The first is the introduction of "real-time" features such as high-resolution timers, priority inheritance, and shortened non-preemptable sections in mainline Linux (in versions 2.6.16 to 2.6.22). The second is the introduction (in version 2.6.24) of mechanisms for supporting *container hierarchies* (LVS, 2007; Eriksson and Palmroos, 2007; Lessard, 2003). Containers are an abstraction that allows different task groups to be isolated from one another (mainly, by providing different name spaces to different task groups for referring to tasks, files, *etc.*). Containers are seen as a lightweight way to achieve many of the benefits provided by virtualization, without the expense of hosting multiple operating systems. From the standpoint of scheduling, containers are similar to various "server" abstractions considered in the real-time-systems literature.

These Linux-related developments are happening at a time when multiprocessor platforms are becoming increasingly common. This is partly due to the advent of multicore technologies as an alternative to single-core chip designs. Additionally, reasonably-priced "server class" multiprocessors have been available for some time now. These hardware-related developments are profound, because they mean that multiprocessors are now a "common-case" platform that software designers must deal with.

Motivated by these trends, we consider in this paper the problem of efficiently scheduling arbitrary real-time container hierarchies on a multiprocessor platform. Unlike most prior related efforts (see below), we are mainly interested in supporting *soft* timing constraints. This is partly due to our

*This is an extended version of the ECRTS paper (Leontyev and Anderson, 2008b).

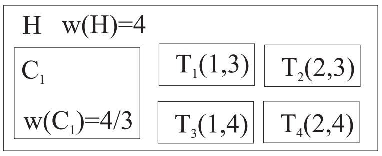


Figure 1: A host container H that encapsulates another container C_1 and four real-time tasks $T_1, \ldots, T_4$. Some of the notation used in this figure is explained in later sections.

interest in Linux: while there is much interest in using Linux to support soft real-time workloads, Linux is not a real-time operating system and thus cannot be used to support "true" hard timing constraints. In addition, there is growing awareness in the real-time-systems community that, in many settings, soft constraints are far more common than hard constraints (Rajkumar, 2006). If hard constraints do exist, then ensuring them *efficiently* on most multiprocessor platforms is problematic anyway, due to a lack of effective timing-analysis tools for determining task execution costs. (While timing analysis is needed for soft real-time systems as well, less-accurate empirically-derived costs often suffice in such systems.)

The problem. For our purposes, a *container* is a scheduling abstraction. Containers are organized hierarchically in a tree. A container may have as children other containers or tasks, as seen in the example in Figure 1. (In Linux, the container hierarchy may change dynamically; we defer consideration of dynamic changes to future work.) Each real-time task is assumed to be sporadic (see Section 2) and is either hard or soft: hard tasks cannot miss their deadlines, while soft tasks can. However, misses in the latter case may be by bounded amounts only. Associated with each container is a specified *bandwidth*, which denotes the fraction of the overall (multiprocessor) system's capacity to which it is entitled. When a container receives processor time, it allocates that time to one of its children. Our goal is to devise a scheme for performing such allocations throughout the container hierarchy. Although we do allow for the presence of hard real-time tasks, we implicitly assume that they are few in number. That is, our main objective is to ensure that allocations are performed efficiently when most (or all) tasks are soft.

Of course, one way to meet this objective is by simply viewing all timing constraints as hard. However, in a container hierarchy, this will result in significant utilization loss. In particular, the schedulability of the tasks within a container depends on the processing capacity allotted to that container—the *supply*—and the processing capacity required by the tasks within the container—the *demand*. Very loosely speaking, verifying schedulability involves showing that demand (over some time interval of interest) cannot exceed supply. When timing constraints are hard, supply and demand are characterized using functions that cause supply to be under-estimated and demand to be over-estimated. The net effect is that, at each level of a container hierarchy, some non-negligible amount of overall utilization is lost. The deeper the container hierarchy, the greater the loss. In fact, the overall loss can be so great, unrestricted hierarchical containment simply becomes untenable.

Prior work. As noted earlier, the notion of a "container" as considered in this paper is more commonly called a "server" in the real-time-systems literature. Server-based abstractions were first considered in the context of uniprocessor systems, and a number of schemes intended for such systems have been proposed (many of them can be found in (Liu, 2000)). Several multiprocessor schemes that are extensions of prior uniprocessor schemes have also been proposed (Baruah et al, 2002; Baruah and Lipari, 2004; Pellizzoni and Caccamo, 2008). However, in all of these schemes, it is assumed that all task deadlines are hard. Systems that may also have tasks with soft deadlines have been considered very recently (Brandenburg and Anderson, 2007). In addition, several Pfair-based multiprocessor server schemes have been proposed, again, mostly for systems with only hard deadlines (Anderson et al, 2006; Holman and Anderson, 2006; Moir and Ramamurthy, 1999; Srinivasan et al, 2002). In all of the work cited so far, only two-level container hierarchies are considered. Moreover, the Pfair-based schemes just cited are subject to higher runtime overheads than other schemes, due to the fact that Pfair scheduling algorithms may preempt and migrate tasks often. The only prior work of which we are aware in which multi-level container hierarchies are considered on multiprocessor platforms is a recent paper by Shin et al. (Shin et al, 2008). However, as with most other prior work, only hard deadlines are considered in their paper. To the best of our knowledge, soft deadlines have not been considered before in scheduling-related research on supporting multi-level container hierarchies on multiprocessors.

In the approach of Shin et al. (Shin et al, 2008), the global earliest-deadline-first algorithm (GEDF) is used as the per-container scheduler. Under GEDF, tasks are scheduled from a single run queue and their jobs are prioritized on an earliest-deadline-first (EDF) basis. One interesting property of GEDF is that, under it, bounded deadline tardiness can be ensured for sporadic tasks without severely constraining overall utilization (Devi and Anderson, 2008). In recent work, it has been shown that the same is true for a wider class of global scheduling algorithms (Leontyev and Anderson, 2008a). In this paper, we exploit these results to obtain a hierarchical scheme in which deadline tardiness is bounded for soft real-time tasks.

Contributions. Our main contribution is a new multiprocessor scheduling approach for multi-level container hierarchies in which both hard and soft sporadic real-time tasks can be supported. With hard real-time tasks, some utilization loss is incurred (which seems inevitable). However, in a system with only soft real-time tasks, no utilization loss is incurred (assuming that system overheads are negligible—such overheads will cause some loss in any scheme in practice). This statement is true, provided the goal is to schedule soft real-time tasks so that their tardiness is bounded, no matter how great the bound may be. In addition to presenting our overall scheme, we also present the results of experiments conducted to assess its usefulness. In these experiments, our scheme exhibited performance—in terms of both necessary processing capacity and tardiness—comparable to that of schemes that exhibit good performance but are oblivious to containers (and hence, do not provide any container isolation).

The rest of this paper is organized as follows. In Section 2, we present our system model. In Section 3, we formally characterize the "supply" available to a container and propose a container scheduling scheme. In Sections 4 and 5, we present methods for checking the schedulability of real-time tasks within a container and for computing the supply available to its child containers (if any). In Section 6, we discuss tradeoffs pertaining to having hard real-time tasks in containers. In Section 7, we examine the extent to which temporal isolation is ensured in container hierarchies under our scheduling scheme. In Section 8, we present our experimental results. We conclude in Section 9.

2 System Model

In order to support the scheduling of containers within an arbitrary hierarchy, it suffices to consider the problem of scheduling a single container H on a set of $M(H)$ unit-speed processors, where some processors may not be available for execution during certain time intervals. The set of child containers and real-time tasks encapsulated in H is referred to as $succ(H)$. (Non-real-time tasks could be contained as well, but we do not consider such tasks in this paper.) At any time, the container may be scheduled on several available processors. When the container is scheduled, some of its children are selected for execution using some internal scheduling policy.

2.1 Sporadic Task Model

The set of real-time tasks encapsulated in the container H is denoted $\tau = \{T_1, \ldots, T_n\}$. In this paper, we assume such tasks are sporadic. Each sporadic task is invoked or *released* repeatedly, with each such invocation called a *job*. Associated with each such task T_i are two parameters, e_i and p_i: e_i gives the maximum *execution time* of one job of T_i, while, p_i, called the *period* of T_i, gives the minimum time between consecutive job releases of T_i. For brevity, we often use the notation $T_i = (e_i, p_i)$ to specify task parameters. The *utilization of task* T_i is defined as $u_i = e_i/p_i$, and the *utilization of the task system* τ as

$$U_{sum}(\tau) = \sum_{T_i \in \tau} u_i. \tag{1}$$

The j^{th} job of task T_i, where $j \geq 1$, is denoted $T_{i,j}$. A task's first job may be released at any time $t \geq 0$. The release time of job $T_{i,j}$ is denoted $r_{i,j}$ and its (absolute) deadline $d_{i,j}$ is defined as $r_{i,j} + p_i$ (implicit deadlines). 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. When a job of a task misses its deadline, the release time of the next job of that task is not altered. However, at most one job of a task may execute at any time, even if deadlines are missed. We call an unfinished job $T_{i,j}$ *ready* at time t if $t \geq r_{i,j}$ and the predecessor job $T_{i,j-1}$ (if any) has completed execution by time t. Job $T_{i,j}$ cannot commence execution before it is ready.

Given these assumptions, if a task has bounded deadline tardiness, then its long-term allocation is proportional to its utilization, as shown later. For *hard* real-time (HRT) tasks, we require that all deadlines are met, while for *soft* real-time (SRT) tasks, we require that deadline tardiness be bounded (regardless of how high the bound may be).

In that which follows, we find it convenient to view a real-time task as a specialized container with no nested children that can be scheduled on at most one processor at any time and that has hard or soft deadlines.

2.2 Container Bandwidth

Each container H is characterized by its *bandwidth* $w(H) \geq 0$, which specifies the processing capacity to which it is entitled. For a real-time task T_i, we define $w(T_i) \triangleq u_i$. Since the containers in $succ(H)$ are scheduled when the parent container is scheduled, their allocation time cannot exceed that of H. Therefore, we require

$$w(H) \geq \sum_{C_j \in succ(H)} w(C_j). \tag{2}$$

Example 1. In Figure 1, a host container H with bandwidth $w(H) = 4$ encapsulates a child container C_1 with bandwidth $w(C_1) = 4/3$, two HRT tasks $T_1(1,3)$ and $T_2(2,3)$, and two SRT tasks $T_3(1,4)$ and $T_4(2,4)$.

Overview of our approach. In the following sections, we solve the problem described at the beginning of Section 2 via a decomposition into two subproblems, each of which can be solved by applying previously-published results. First, we split the bandwidth of each container, parent and child, into integral and fractional parts and argue that the integral parts can easily be dealt with. The fractional part of each child container is then handled by creating a special SRT *server task* with utilization equal to that fractional portion. This leads to our first subproblem, which is that of scheduling within the parent container, using the "supply" available to it, all child HRT and SRT tasks (where some of the SRT tasks may be server tasks). We then deal with any HRT tasks by encapsulating them within a new child container that schedules these tasks on an integral number of processors via a prior HRT scheduling scheme. This leaves us with our second subproblem, which is to schedule within the parent container a collection of SRT tasks. We solve this problem by exploiting prior results on using global scheduling algorithms to ensure bounded tardiness. So that our overall scheme can be applied recursively in a container hierarchy, we finish our analysis by characterizing the supply available to each child container.

3 Container Scheduling

The host container H receives processor time from $M(H)$ individual processors. We now further constrain the manner in which any container C receives processor time by assuming the following.

(P) At any time, a container C can be scheduled on $m(C) \triangleq \lfloor w(C) \rfloor$ or $M(C) \triangleq \lceil w(C) \rceil$ processors.

This restriction minimizes the execution parallelism available to C so that, for any interval of length Δ, C's allocation is within $[\lfloor w(C) \rfloor \Delta, \lceil w(C) \rceil \Delta]$. For real-time tasks, this restriction holds implicitly, because a real-time task T_i is scheduled on at most one processor at any time and $w(T_i) = u_i \leq 1$, so $\lceil w(T_i) \rceil = 1$ and $\lfloor w(T_i) \rfloor = 0$. We say that a processor is *fully available* to C, if it is dedicated exclusively to C. Given Restriction (P), we can assume that $m(C)$ processors are fully available to C.

As explained in detail later, there are two reasons for introducing Restriction (P). First, increasing the amount of supply parallelism (the number of available processors) restricts the maximum per-task utilization and the total system utilization if the long-term supply remains fixed. Second, maximizing the number of processors fully available to C lessens deadline tardiness for any child real-time task. Intuitively, this is because such tasks are *sequential* and thus may leave processors unused if parallelism is increased too much.

Example 2. Consider a container H with bandwidth $w(H) = 4/3$ that encapsulates a task $T_1(5,6)$, as shown in Figure 2(a). Suppose that processor time is supplied as shown in Figure 2(b) so that

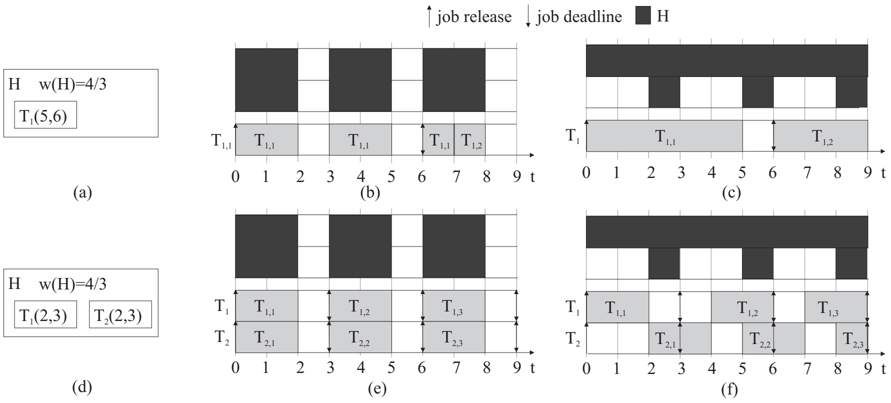


Figure 2: Comparison of supply parallelism in Examples 2 and 3.

H occupies two processors for two time units every three time units. The supply available to H is approximately $\frac{4 \cdot \Delta}{3}$ for any sufficiently long interval Δ. However, H does not execute during the interval $[2, 3)$, so Restriction (P) is violated, because $\lfloor w(H) \rfloor = \lfloor 4/3 \rfloor = 1$. Task T_1's jobs demand five execution units every six time units, but because they must execute sequentially, they can execute for only four time units every six time units. Thus, task T_1's tardiness can be unbounded. In the schedule in Figure 2(c), container H also receives four execution units every three time units, but in contrast to Figure 2(b), Restriction (P) is satisfied. Because one processor is fully available to H, task T_1 meets all of its deadlines.

As one may suspect, enforcing Restriction (P) may sometimes have negative consequences. Indeed, a task set with a large number of tasks may benefit from a larger number of available processors if all deadlines have to be met.

Example 3. Consider the container H from the previous example, except that it now encapsulates two real-time tasks $T_1(2, 3)$ and $T_2(2, 3)$, as shown in Figure 2(d). In the schedule shown in Figure 2(e), which is equivalent from the container's perspective to that in Figure 2(b), jobs of T_1 and T_2 meet their deadlines. However, in the schedule in Figure 2(f), where Restriction (P) is enforced as in Figure 2(c), job $T_{2,1}$ misses its deadline at time 3 because it cannot execute on two processors simultaneously during the time interval $[2, 3)$. Still, in this schedule, T_2's tardiness is only one time unit.

The two examples above illustrate that, while minimizing supply parallelism may negatively impact timeliness, it allows the widest range of loads to be scheduled with bounded deadline tardiness, which is in accordance with our focus on SRT tasks. In (Shin et al, 2008), mentioned earlier, the objective is instead to schedule HRT tasks. There, the alternative approach of maximizing supply parallelism is used.

We now develop a scheduling policy that enforces Restriction (P) for child containers assuming that it holds for the host container H. Given the latter, H is supplied time from $M(H)$ processors, where $m(H)$ processors are always available for scheduling $succ(H)$ and at most one processor is partially available.

A child container $C_i \in succ(H)$ must occupy at least $m(C_i)$ processors at any time. By (2), $w(H) \geq \sum_{C_i \in succ(H)} w(C_i)$, and hence, $m(H) = \lfloor w(H) \rfloor \geq \lfloor \sum_{C_i \in succ(H)} w(C_i) \rfloor \geq \sum_{C_i \in succ(H)} \lfloor w(C_i) \rfloor = \sum_{C_i \in succ(H)} m(C_i)$. Therefore, we can make $m(C_i)$ processors fully available to each child container $C_i \in succ(H)$ by using the $m(H)$ processors fully available to H. Note that, for containers with $w(C_i) < 1$ (including real-time tasks), $m(C_i) = \lfloor w(C_i) \rfloor = 0$. In any event, given this design decision, each child container C_i receives at least $m(C_i)\Delta$ units of time over an interval of length Δ.

If a child container C_i is not a real-time task and $m(C_i) < w(C_i)$, then it occasionally needs supply from an additional processor. For this, we construct a SRT periodic *server task* $S_i(e_i, p_i)$, where $u_i = e_i/p_i = w(C_i) - m(C_i) < 1$. (The term *periodic* means that $r_{i,j} = (j-1) \cdot p_i$ holds for each $j \geq 1$.)

We denote the set of server tasks as $\tau^S = \{S_1, \ldots, S_n\}$. Jobs of these tasks are scheduled together with the jobs of encapsulated real-time tasks using the remaining $m(H) - \sum_{C_j \in succ(H)} m(C_j)$ fully available processors and at most one partially available processor. When task S_i's jobs are scheduled, an additional processor is available to container C_i. Because server task S_i is constructed only if $w(C_i) > m(C_i) = \lfloor w(C_i) \rfloor$, we have $\lceil w(C_i) \rceil = m(C_i) + 1 = M(C_i)$. Thus, container C_i always occupies $m(C_i)$ processors, and $M(C_i)$ processors are occupied when a job of S_i is scheduled. Thus, Restriction (P) is ensured for each child container.

Example 4. Consider container H from Example 1. For container C_1, one processor is reserved because $\lfloor w(C_1) \rfloor = \lfloor 4/3 \rfloor = 1$. For this container, we also construct a SRT server task $S_1(1, 3)$, so that $\lfloor w(C_1) \rfloor + e_1/p_1 = 1 + 1/3 = w(C_1)$. When jobs of S_1 are scheduled, an additional processor is available to container C_1, as shown in Figure 3(b).

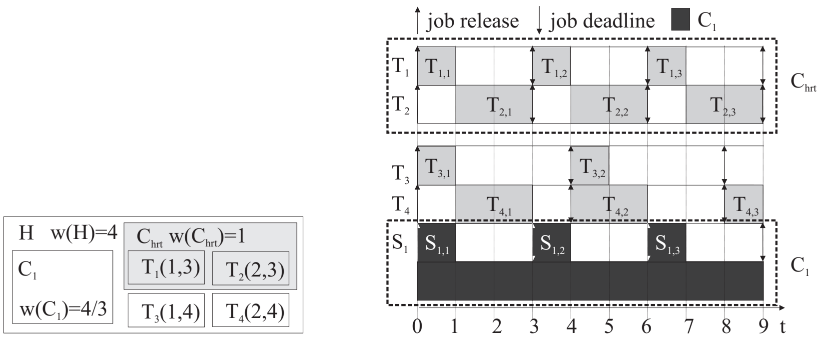
Let $\mathsf{HRT}(H)$ (respectively, $\mathsf{SRT}(H)$) be the set of HRT (respectively, SRT) tasks encapsulated in H. The remaining problem at hand, referred to as Subproblem 1, is that of scheduling tasks from the sets $\mathsf{HRT}(H)$, $\mathsf{SRT}(H)$, and τ^S on some number of fully available processors and at most one partially available processor.

4 Subproblem 1

To schedule the tasks in $\mathsf{HRT}(H)$, we encapsulate them into a child container C_{hrt} with integral bandwidth $w(C_{hrt}) = m(C_{hrt}) = M(C_{hrt})$. Applying Restriction (P) to C_{hrt}, $m(C_{hrt})$ processors must be reserved for this container. In this section, we consider two approaches for scheduling the remaining tasks in $\mathsf{SRT}(H)$ and τ^S; in the first approach, HRT and SRT tasks do not execute on the same processors, and in the second approach, they may.

Basic approach. The tasks in $\mathsf{HRT}(H)$ can be scheduled within C_{hrt} using a variety of approaches. Given our emphasis on SRT tasks, we simply use the partitioned EDF (PEDF) algorithm for this purpose, deferring consideration of other approaches to future work. Under PEDF, tasks are statically assigned to processors and each processor schedules its assigned tasks independently on an EDF basis. Assume that processor h is among the $m(C_{hrt})$ processors reserved for container C_{hrt} and let τ_h denote the set of sporadic HRT tasks assigned to that processor. All task deadlines will be met on processor h if

$$U_{sum}(\tau_h) = \sum_{T_i \in \tau_h} u_i \leq 1, \tag{3}$$



(a) (b)

Figure 3: Example 5. **(a)** Isolating HRT tasks. **(b)** A schedule with the two HRT tasks in a separate container.

which is a well-known uniprocessor EDF schedulability test (Liu and Layland, 1973). This test, when applied in a multiprocessor system, presumes a given assignment of tasks to processors. Such an assignment (and correspondingly, the number of processors required for C_{hrt}) can be determined using any of various bin-packing heuristics. Further results concerning PEDF schedulability tests can be found in (Baruah and Fisher, 2005; Chakraborty and Thiele, 2005; Liu, 2000).

As mentioned earlier, HRT policies may introduce utilization loss. For PEDF, there exist task sets, for which the reserved processors could be underutilized. However, if HRT tasks are relatively few in number, such loss will likely be small, compared to the total utilization of SRT tasks. Loss is incurred when creating C_{hrt} if its bandwidth (given by the number of processors required for it) exceeds the sum of the utilizations of the HRT tasks it contains. If this is the case, then (2) must be validated with the tasks in $\mathsf{HRT}(H)$ replaced by the container C_{hrt}.

Example 5. Consider again container H from Example 1. In our approach, we encapsulate the two HRT tasks $T_1(1,3)$ and $T_2(2,3)$ into a container C_{hrt}, as shown in Figure 3(a). The total utilization of these two tasks is $U_{sum} = u_1 + u_2 = 1/3 + 2/3 = 1$. By (3), these two tasks will meet their deadlines if scheduled using uniprocessor EDF. We set $w(C_{hrt}) = 1$, so the container C_{hrt} will require one processor. The total bandwidth of container H's children is $\sum_{C_i \in succ(H)} w(C_i) = w(C_1) + w(C_{hrt}) + w(T_3) + w(T_4) = 4/3 + 1 + 1/4 + 2/4 = 37/12 < 4 = w(H)$, so (2) is satisfied. When scheduling the modified container H on $\lceil w(H) \rceil = 4$ processors, as shown in Figure 3(b), one processor is reserved for the HRT container C_{hrt} and tasks T_1 and T_2 are scheduled on that processor. Note that no utilization loss is incurred by HRT tasks. In Example 4, we reserved one processor for container C_1 and constructed the server task $S_1(1,3)$. Jobs of this server task are scheduled with the jobs of tasks T_3 and T_4 on the two remaining fully available processors.

Note that, if a system has a small number of processors, then it may not be possible to dedicate an integral number of processors for a HRT container as described above. For example, if the parent container H has fractional bandwidth, then its encapsulated HRT tasks may be required to execute on a partially available processor, which would require different analysis from that in this paper. Given our focus on SRT tasks, such analysis is beyond the scope of this paper. However, if a system is purely SRT, an arbitrarily deep hierarchy of SRT containers can be maintained even in the uniprocessor case.

In the case when it is possible to reserve an integral number of processors for HRT tasks, it may not be possible to accommodate SRT tasks using the remaining bandwidth as the following example illustrates.

Example 6. Consider Figure 4(a), which depicts a container H that is similar to that from Example 1, except that T_2 has a smaller execution time and there are two additional SRT tasks, $T_5(1,2)$ and $T_6(1,2)$. In our approach, we encapsulate the two HRT tasks $T_1(1,3)$ and $T_2(1,3)$ into a container C_{hrt}, as shown in Figure 4(a). The total utilization of these two tasks is $U_{sum} = u_1 + u_2 = 1/3 + 1/3 = 2/3$. By (3), these two tasks will meet their deadlines if scheduled using uniprocessor EDF. We set $w(C_{hrt}) = 1$, so the container C_{hrt} requires one processor. When scheduling the modified container H on $\lceil w(H) \rceil = 4$ processors, as shown in Figure 4(b), one processor is reserved for the HRT container C_{hrt}, and tasks T_1 and T_2 are scheduled on that processor (inset (c) is considered later). As in Example 4, we reserve one processor for container C_1 and construct a server task $S_1(1,3)$. Jobs of this server task are scheduled with the jobs of tasks $T_3, \ldots, T_6$.

Under the basic approach, the processor time that remains after scheduling T_1 and T_2 is unused (see intervals $[2,3)$ and $[5,6)$ within C_{hrt} in Figure 4(b)). Thus, the bandwidth available to tasks S_1 and $T_3, \ldots, T_6$ is $w(H) - m(C_{hrt}) - m(C_1) = 4 - 1 - 1 = 2$. However, the total bandwidth required by tasks S_1 and $T_3, \ldots, T_6$ is $w(S_1) + w(T_3) + w(T_4) + w(T_5) + w(T_6) = 1/3 + 1/4 + 2/4 + 1/2 + 1/2 = 25/12 > 2$, and hence, tasks S_1 and $T_3, \ldots, T_6$ will have unbounded deadline tardiness. Note that, in the schedule in Figure 4(b), the ready job $T_{6,2}$ is not scheduled during the interval $[2,3)$ even though there is an available processor. Similarly, the ready job $T_{4,2}$ is not scheduled during the interval $[5,6)$.

Extended approach. In order to allocate the available bandwidth more efficiently, we can use the time not allocated to HRT tasks on some of the $m(C_{hrt})$ processors reserved for such tasks to schedule tasks in $\mathsf{SRT}(H) \cup \tau^S$ (in addition to the supplied time on other processors). We allow this approach to be selectively applied by defining the parameter $K(H)$ below.

Definition 1. Let $K(H) \in [0, m(C_{hrt})]$ be the number of processors where tasks in $\mathsf{HRT}(H)$ and $\mathsf{SRT}(H) \cup \tau^S$ are co-scheduled.

We assume that HRT tasks are statically prioritized over SRT and server tasks. Thus, HRT tasks still execute as if an integral number of processors were dedicated to their exclusive use. After assigning all HRT tasks to the $m(C_{hrt})$ processors reserved for them and then selecting $K(H)$, the utilization loss due to partitioning is $U_{\mathsf{lost}} = \sum_{k=K(H)+1}^{m(C_{hrt})} (1 - U_{sum}(\tau_k))$ (we assume that HRT-allocated processors are numbered in order of increasing $U_{sum}(\tau_k)$). Though engaging additional processors for scheduling tasks in $\mathsf{SRT}(H) \cup \tau^S$ (i.e., increasing $K(H)$) reduces utilization loss and sometimes is imperative in order to accommodate all SRT tasks, a large value for $K(H)$ may negatively impact SRT schedulability as discussed later in Section 5; tradeoffs involved in selecting $K(H)$ are discussed in Section 6. After weighing such tradeoffs and selecting a value for $K(H)$, (4) below must be validated to account for any lost bandwidth.

$$w(H) \geq \sum_{C_j \in succ(H)} w(C_j) + U_{\mathsf{lost}} \tag{4}$$

Example 7. Consider container H from Example 6. A schedule where HRT processor time is reclaimed (i.e., $K(H) = 1$) is shown in Figure 4(c). The bandwidth available to tasks S_1 and

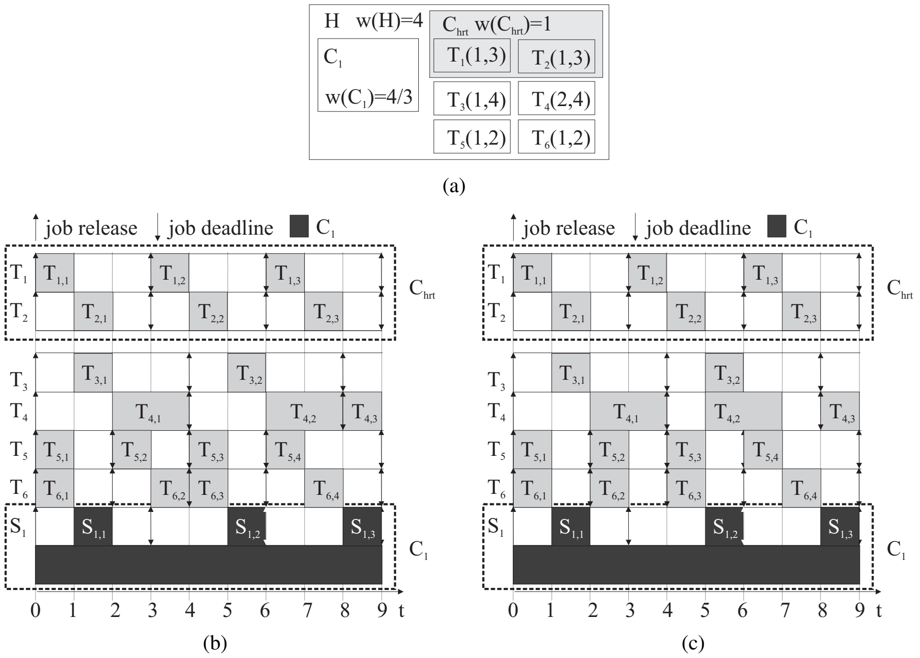


Figure 4: **(a)** Container considered in Examples 6 and 7. A schedule **(b)** with and **(c)** without HRT time reclamation.

$T_3, \ldots, T_6$ is $w(H) - w(T_1) - w(T_2) - m(C_1) = 4 - 1/3 - 1/3 - 1 = 7/3$, which is greater than the bandwidth required by these tasks. Note that, in this schedule, the processors supplied to H are idle only if there are not enough ready tasks to occupy all of them.

Having dispensed with any HRT tasks, we can complete our solution to Subproblem 1 by devising a scheduling policy that ensures bounded tardiness for the remaining SRT tasks, some of which may be server tasks.

Definition 2. **(τ_s, M_s, and Subproblem 2)** Let $\tau_s = \mathsf{SRT}(H) \cup \tau^S$. These tasks are to be scheduled on M_s processors, of which $m(H) - \sum_{C_j \in succ(H)} m(C_j) - m(C_{hrt})$ are fully available and $K(H) + G$, where $G \leq 1$, are partially available. Note that $K(H)$ processors are partially available due to HRT tasks internal to H and at most one additional processor is partially available because the supply provided by H's parent is subject to Restriction (P).

We refer to this last remaining subproblem as Subproblem 2.

5 Subproblem 2

In solving Subproblem 2, restrictions on supplied processor time are of relevance. Such restrictions can be dealt with using per-processor *supply or availability functions* (Chakraborty and Thiele, 2005).

Definition 3. (supply functions) The supply (or availability) function $\beta_k^l(\Delta) : R \rightarrow R$, provides a lower bound on the amount of processor time processor k can guarantee during any time interval of length Δ. This function is defined as

$$\beta_k^l(\Delta) = \max(0, \widehat{u_k} \cdot (\Delta - \sigma_k)), \quad (5)$$

where $\widehat{u_k} \in (0, 1]$ and $\sigma_k \geq 0$. $\widehat{u_k}$ is called the *processor bandwidth* and σ_k the *maximum blackout time*, because σ_k is the maximum interval when processor k may not provide any supply (Easwaran et al, 2007).

The following property is a straightforward application of the above definition.

(**FA**) If processor k is fully available, then $\beta_k^l(\Delta) = \Delta$, $\widehat{u_k} = 1$, and $\sigma_k = 0$.

From the earlier statement of Subproblem 2 in Definition 2, of the M_s processors under consideration, $K(H) + G$ are partially available. We assume that these M_s processors are indexed so that the supply from them can be described using M_s supply functions: $\beta_k^l(\Delta) = \max(0, \widehat{u_k}(\Delta - \sigma_k))$, where $0 < \widehat{u_k} \leq 1$ and $\sigma_k \geq 0$, for $1 \leq k \leq K(H) + G$; and $\beta_k^l(\Delta) = \Delta$, for $K(H) + G + 1 \leq k \leq M_s$. If $K(H) + G \leq 1$, i.e., at most one processor is partially available, then we say that such a collection of functions is in *Minimum Parallelism* (MP) form. As explained later, ensuring that supply is in MP form allows the widest range of SRT workloads to be supported without incurring utilization loss.

Before continuing, note that if $M_s = 1$, i.e., all remaining SRT tasks are to be scheduled on one processor, then EDF can be used on that processor. If this processor is fully available, then tardiness will be zero for these tasks (due to the optimality of EDF), and if it is partially available, then it can be easily shown to be bounded, using real-time calculus (Chakraborty and Thiele, 2005), provided $U_{sum}(\mathsf{SRT}(H) \cup \tau^S) \leq \widehat{u_1}$. In the remainder of this section, we concentrate on the more interesting case, $M_s \geq 2$. In this case, our approach leverages some recent theoretical results, which we describe next.

5.1 Window-Constrained Scheduling

The problem of scheduling a set of sporadic SRT tasks on multiple processors with restricted supply was considered in (Leontyev and Anderson, 2008a). In this work, a class of global scheduling policies that ensure bounded deadline tardiness was considered. This class of algorithms is described next.

Let τ be a set of sporadic SRT tasks scheduled on $M \geq 2$ processors, with supply functions $\beta_k^l(\Delta) = \max(0, \widehat{u_k}(\Delta - \sigma_k))$, where $1 \leq k \leq M$. (Note that τ was defined earlier in Section 2.1. Here, we mean τ to denote any sporadic SRT task set. The distinction should be clear from the context.) Assume

$$U_{sum}(\tau) \leq \sum_{k=1}^{M} \widehat{u_k}, \quad (6)$$

i.e., the total system utilization is at most the total supplied bandwidth. Released jobs are placed into a single global ready queue. When choosing a new job to schedule, the scheduler selects (and dequeues) the ready job of highest priority. Priorities are determined as follows assuming that any ties are broken arbitrarily but consistently.

Definition 4. (prioritization functions) Associated with each released job $T_{i,j}$ is a function of time $\chi(T_{i,j}, t)$, called its *prioritization function*. If $\chi(T_{i,j}, t) < \chi(T_{k,h}, t)$, then the priority of $T_{i,j}$ is higher than the priority of $T_{k,h}$ at time t.

Definition 5. (window-constrained priorities) A scheduling algorithm's prioritization functions are *window-constrained* iff, for each task T_i, there exist constants ϕ_i and ψ_i such that, for each job $T_{i,j}$ of T_i and time t,

$$r_{i,j} - \phi_i \leq \chi(T_{i,j}, t) \leq d_{i,j} + \psi_i. \tag{7}$$

Note that (7) requires a job's χ-values to lie within a window $[r_{i,j} - \phi_i, d_{i,j} + \psi_i]$ that is defined with respect to its release time and deadline. Note also that the constants ϕ_i and ψ_i may be positive or negative; however, if negative, the interval $[r_{i,j} - \phi_i, d_{i,j} + \psi_i]$ cannot be empty.

GEDF is an example of a global algorithm with window-constrained priorities. Under it, $\chi(T_{i,j}, t) = d_{i,j}$ for each job $T_{i,j}$. In (Leontyev and Anderson, 2008a), a tardiness bound is established that applies to any window-constrained global scheduling algorithm. To state this bound, let

$$\rho = \max\left(0, \max_{i \neq a}(\phi_i + \psi_a)\right) \quad \text{and} \quad \mu = \max\left(0, \max_{i \neq a}(\psi_a + p_a + \phi_i)\right). \tag{8}$$

Further, let $U(\tau, y)$ be the set of at most $\min(|\tau|, y)$ tasks from τ of highest utilization, and let

$$U_L = \sum_{T_i \in U(\tau, M-1)} u_i. \tag{9}$$

Similarly, let $E(\tau, y)$ be the set of at most $\min(|\tau|, y)$ tasks from τ of highest execution cost, and let

$$E_L = \sum_{T_i \in E(\tau, M-1)} e_i.$$

Finally, let F be the number of processors with $\beta_k^l(\Delta) \neq \Delta$, and let $x = \max(\rho, z)$, where

$$z = \frac{E_L + \max\limits_{1 \leq \ell \leq |\tau|}(A(\ell))}{\sum_{k=1}^{M} \widehat{u_k} - \max(F-1, 0) \cdot \max\limits_{1 \leq \ell \leq |\tau|}(u_\ell) - U_L}, \tag{10}$$

and

$$\begin{aligned} A(\ell) = e_\ell \cdot \left(\sum_{k=1}^{M}(1 - \widehat{u_k}) - 1\right) + 2\sum_{k=1}^{M} \widehat{u_k} \cdot \sigma_k \\ + \sum_{T_i \in \tau \setminus T_\ell} \left(\left\lceil \frac{\rho + \mu}{p_i} \right\rceil + 1\right) \cdot e_i + \min(M - F, M - 1) \cdot \rho. \end{aligned} \tag{11}$$

Theorem 1. (Proved in (Leontyev and Anderson, 2008a)) *The tardiness of any task $T_k \in \tau$ under a window-constrained scheduling algorithm $\mathcal{A}$ is at most $x + e_k$, where x is as defined above, provided the denominator of (10) is positive.*

5.2 Minimizing the Tardiness Bound

Given the theorem stated above, we now argue in favor of Restriction (P) and show how enforcing this restriction affects the tardiness bound in Theorem 1. Consider the denominator of (10):

$$\sum_{k=1}^{M} \widehat{u_k} - \max(F-1,0) \cdot \max_{1 \le \ell \le |\tau|}(u_\ell) - U_L. \tag{12}$$

The requirement for (12) to be positive implicitly restricts the maximum per-task utilization if $F > 1$, i.e., if two or more processors are partially available. Note also that the value of x is minimized if (12) is maximized. Suppose that the total supplied bandwidth $W = \sum_{k=1}^{M} \widehat{u_k}$ is fixed. Then, (12) will be maximized if either $\max(F-1,0) \cdot \max_{1 \le \ell \le |\tau|}(u_\ell)$ or U_L or both are minimized. The value of U_L depends exclusively on task utilizations and the total number of processors M, as (9) suggests. Therefore, U_L will be minimized if the total number of processors M is minimized. The expression $\max(F-1,0) \cdot \max_{1 \le \ell \le |\tau|}(u_\ell)$ is minimized if $F \le 1$, that is, at most one processor is partially available. Thus, if the total processor bandwidth W is fixed, then (12) is maximized by setting $M = \lceil W \rceil$ and having $\lfloor W \rfloor$ processors fully available. The bandwidth of at most one partially available processor (if any) is $\widehat{u_1} = W - \lfloor W \rfloor$.

The above discussion suggests that bounded tardiness among SRT and server tasks can be achieved for the widest range of task utilizations if the supply to $\mathsf{SRT}(H) \cup \tau^S$ is given in MP form. This is the case if either $K(H) = 0$, (e.g., when $\mathsf{HRT}(H) = \emptyset$ or no spare HRT capacity is reused) or $G = 0$ and $K(H) \le 1$ (i.e., when the bandwidth supplied to H is integral and HRT capacity is reused on at most one processor). If $K(H) + G > 1$, then bounded tardiness may be guaranteed for certain SRT workloads. Various tradeoffs are possible with regard to the selection of $K(H)$. These tradeoffs are discussed in Section 6. After applying Theorem 1 to Subproblem 2, we have the following.

Corollary 1. *Let τ_s, M_s, $K(H)$, and G be as defined in Definition 2. The tardiness of any task $T_k \in \tau_s$ under a window-constrained scheduling policy is at most* $\max(z, \rho) + e_k$*, where*

$$z = \frac{E_L + \max_{1 \le \ell \le |\tau_s|}(A(\ell))}{M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} - \max(K(H)+G-1,0) \max_{1 \le \ell \le |\tau_s|}(u_\ell) - U_L}, \tag{13}$$

$$\begin{aligned} A(\ell) = {} & e_\ell \cdot \left(\sum_{k=1}^{K(H)+G} (1 - \widehat{u_k}) - 1 \right) + 2 \cdot \sum_{k=1}^{K(H)+G} \widehat{u_k} \cdot \sigma_k \\ & + \sum_{T_k \in \tau_s \setminus T_\ell} \left(\left\lceil \frac{\rho + \mu}{p_k} \right\rceil + 1 \right) \cdot e_k + \min(M_s - K(H) - G, M_s - 1) \cdot \rho \end{aligned}$$

provided (6) holds (with M replaced with M_s and τ replaced with τ_s) and (14) below holds.

$$M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} - \max(K(H)+G-1,0) \max_{1 \le \ell \le |\tau_s|}(u_\ell) - U_L > 0 \tag{14}$$

Proof. We prove the corollary using results from Section 5.1; henceforth, when such results are applied, we assume that M is replaced with M_s and τ is replaced with τ_s. In the formulation of Subproblem 2, $K(H)+G$ supply functions $\beta_1^l(\Delta)$ may differ from Δ. Thus, $F = K(H)+G$. By **(FA)**,

$$(\forall k : K(H)+G+1 \leq k \leq M_s :: \sigma_k = 0 \wedge \widehat{u_k} = 1). \tag{15}$$

Thus,

$$\sum_{h=1}^{M_s} \widehat{u_h} = \sum_{h=1}^{K(H)+G} \widehat{u_h} + \sum_{h=K(H)+G+1}^{M_s} \widehat{u_h} = \sum_{h=1}^{K(H)+G} \widehat{u_h} + (M_s - K(H) - G), \tag{16}$$

$$\left(\sum_{k=1}^{M_s}(1-\widehat{u_k}) - 1\right) = \left(\sum_{k=1}^{K(H)+G}(1-\widehat{u_k}) - 1\right), \tag{17}$$

and

$$2 \cdot \sum_{k=1}^{M_s} \widehat{u_k} \cdot \sigma_k = 2 \cdot \sum_{k=1}^{K(H)+G} \widehat{u_k} \cdot \sigma_k. \tag{18}$$

Setting $F = K(H)+G$ and substituting (17) and (18) into (11), we get $A(\ell)$ as defined in the statement of the corollary. Finally, substituting (16) into (10), we get z as defined in the statement of the corollary. □

If GEDF is used for SRT tasks, then the tardiness bound in Corollary 1 can be further tightened by setting $A(\ell)$ in (13) to $e_\ell \cdot (\sum_{k=1}^{K(H)+G}(1-\widehat{u_k})-1)+2\cdot\sum_{k=1}^{K(H)+G}\widehat{u_k}\cdot\sigma_k$, as shown in (Leontyev and Anderson, 2008a).

The following lemma shows that providing supply in MP form allows the widest range of SRT workloads to be supported.

Lemma 1. *If the supply to the tasks in τ_s is in MP form, then (14) always holds.*

Proof. If the supply to τ_s is in MP form, then $K(H)+G \leq 1$. We thus have

$$M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} = M_s - 1 + \widehat{u_1}. \tag{19}$$

Setting $K(H)+G \leq 1$ and (19) into the LHS of (14) we have

$$\begin{aligned} M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} - \max(K(H)+G-1, 0) \max_{1\leq\ell\leq|\tau_s|}(u_\ell) - U_L \\ = M_s - 1 + \widehat{u_1} - U_L. \end{aligned} \tag{20}$$

We now consider two cases depending on the number of tasks in τ_s.

Case 1: $|\tau_s| \leq M_s - 1$. In this case, by (9),

$$U_L = \sum_{i=1}^{|\tau_s|} u_i \leq M_s - 1 < M_s - 1 + \widehat{u_1},$$

where the latter inequality follows from Definition 3.

Case 2: $|\tau_s| > M_s - 1$. In this case, by (9),

$$U_L < U_{sum}(\tau_s) \overset{\{\text{by } (6)\}}{\leq} \sum_{h=1}^{M_s} \widehat{u_h} = M_s - 1 + \widehat{u_1},$$

where the latter equality follows from (15). The required result follows from (20) and the two cases above. □

Corollary 2. *If at most one processor is partially available to τ_s, then Corollary 1 only requires that (6) holds. That is, bounded tardiness can be ensured with no utilization loss.*

Note that, if all $M_s \geq 2$ processors are fully available, then a HRT GEDF schedulability test (e.g., (Baruah, 2007; Bertogna et al, 2009; Baruah and Baker, 2008)) can be applied to τ before calculating tardiness bounds. If this test passes, then maximum tardiness is zero.

5.3 Computing Next-Level Supply

The remaining issue is to compute the supply of each child container in MP form, so that our analysis can be applied recursively in a container hierarchy. Note that we can do this regardless of whether the basic or extended approach described in Section 4 is used. Ensuring that child-container supplies are in MP form ensures that Property (P) holds for such containers.

If a server task $S_i(e_i, p_i)$ has bounded deadline tardiness, then the total guaranteed long-term supply to container C_i will be proportional to the long-term supply of $m(C_i)$ fully available processors, which can be described by a set of $m(C_i)$ supply functions equal to Δ, plus that of a partially available processor with bandwidth $u_i = e_i/p_i$. We are left with characterizing the processor time that is available to C_i when the server task S_i is scheduled.

The supply guaranteed to the server task S_i will depend on its parameters, e_i and p_i, and its tardiness. The latter depends on the scheduling algorithm used for SRT and server tasks, their parameters, and (if extended approach is used) the amount of supply reclaimed on HRT-occupied processors.

Definition 6. Let $\mathsf{A}(T_i, t_1, t_2, \mathcal{Q})$ be the allocation of task T_i during the interval $[t_1, t_2)$ in the schedule $\mathcal{Q}$. Let $\mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{Q})$ be the allocation of job $T_{i,j}$ during the interval $[t_1, t_2)$ in the schedule $\mathcal{Q}$.

Lemma 2. *Let Θ_i be the maximum deadline tardiness of the server task S_i's jobs in $\mathcal{Q}$. Then, the allocation $\mathsf{A}(S_i, 0, t, \mathcal{Q})$ satisfies the following.*

$$\mathsf{A}(S_i, 0, t, \mathcal{Q}) \leq u_i \cdot t + e_i(1 - u_i) \tag{21}$$

$$\mathsf{A}(S_i, 0, t, \mathcal{Q}) \geq u_i \cdot t - u_i \cdot \Theta_i - e_i(1 - u_i) \tag{22}$$

Proof. We first prove (21). Let $S_{i,k}$ be the latest job of S_i in schedule $\mathcal{Q}$ such that $r_{i,k} \le t$. (Such a job exists because S_i is a periodic server task.) Then, by Definition 6, the allocation of S_i in $[0, t)$ is

$$
\begin{aligned}
&\mathsf{A}(S_i, 0, t, \mathcal{Q}) \\
&\qquad \{\text{because } S_{i,k}\text{'s successors do not execute before } t \text{ in any schedule}\} \\
&\quad \le \mathsf{A}(S_{i,k}, 0, t, \mathcal{Q}) + \sum_{j<k} \mathsf{A}(S_{i,j}, 0, t, \mathcal{Q}) \\
&\qquad \{\text{because the worst-case execution time of } S_i \text{ is } e_i\} \\
&\quad \le \mathsf{A}(S_{i,k}, 0, t, \mathcal{Q}) + \sum_{j<k} e_i \\
&\qquad \{\text{because } S_{i,k} \text{ is not scheduled before } r_{i,k}\} \\
&\quad \le \min(e_i, t - r_{i,k}) + \sum_{j<k} e_i. \qquad (23)
\end{aligned}
$$

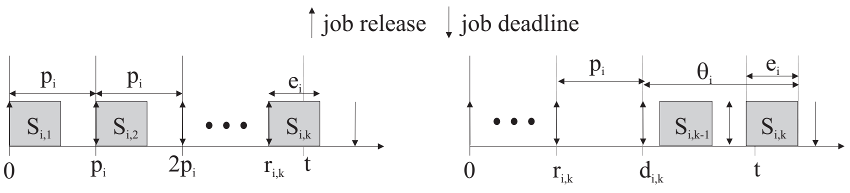
The latter expression is maximized if the number of jobs of S_i released before $r_{i,k}$ is maximized, as shown in Figure 5(a). Therefore, (23) is maximized if $k = \left\lfloor \frac{t}{p_i} \right\rfloor + 1$ and $r_{i,k} = (k-1) \cdot p_i$. Setting these values into (23), we have

$$
\begin{aligned}
&\mathsf{A}(S_i, 0, t, \mathcal{Q}) \\
&\quad \le \min\left(e_i, t - \left\lfloor \frac{t}{p_i} \right\rfloor \cdot p_i\right) + e_i \cdot \left\lfloor \frac{t}{p_i} \right\rfloor \\
&\qquad \{\text{setting } \tfrac{t}{p_i} = q\} \\
&\quad = \min\left(e_i, (q - \lfloor q \rfloor) \cdot p_i\right) + e_i \cdot \lfloor q \rfloor \\
&\quad = \min\left(e_i, (q - \lfloor q \rfloor) \cdot p_i\right) + e_i \cdot \lfloor q \rfloor + e \cdot q - e \cdot q \\
&\quad = \min\left(e_i \cdot (\lfloor q \rfloor - q + 1), (q - \lfloor q \rfloor) \cdot (p_i - e_i)\right) + e_i \cdot q \\
&\qquad \{\text{setting } q - \lfloor q \rfloor = z\} \\
&\quad = \min\left(e_i \cdot (1 - z), z \cdot (p_i - e_i)\right) + e_i \cdot q \\
&\qquad \left\{ \begin{array}{l} \text{the } \min(\dots) \text{ summand is maximized when its two} \\ \text{arguments are equal, which is the case when } z = u_i \end{array} \right\} \\
&\quad \le \min\left(e_i \cdot (1 - u_i), u_i \cdot (p_i - e_i)\right) + e_i \cdot q \\
&\qquad \{\text{setting } q = \tfrac{t}{p_i}\} \\
&\quad = u_i \cdot t + e_i \cdot (1 - u_i).
\end{aligned}
$$

We now prove (22). Let $S_{i,k}$ be the earliest job of S_i such that $d_{i,k} + \Theta_i \ge t$. For this job, since $d_{i,k} = r_{i,k} + p_i$, we have $r_{i,k} + p_i + \Theta_i \ge t$. Let

$$r_{i,k} = t - p_i - \Theta_i + \epsilon, \qquad (24)$$

where $\epsilon \ge 0$. By the selection of $S_{i,k}$, for any job $S_{i,j}$ such that $j < k$, we have $d_{i,j} + \Theta_i < t$. By the statement of the lemma, each job of S_i completes within Θ_i time units after its deadline. Therefore, all jobs $S_{i,j}$ such that $j < k$ complete by time t, i.e.,



(a) (b)

Figure 5: Server task's **(a)** maximum and **(b)** minimum allocation scenarios.

$$\mathsf{A}(S_{i,j}, 0, t, \mathcal{Q}) = e_i \text{ for each } j < k. \tag{25}$$

The allocation $\mathsf{A}(S_{i,k}, 0, t, \mathcal{Q})$ is minimized if $\mathsf{A}(S_{i,k}, t, d_{i,k} + \Theta_i, \mathcal{Q})$ is maximized. The latter is at most $\min(e_i, d_{i,k} + \Theta_i - t)$, as illustrated in Figure 5(b). Thus,

$$\begin{aligned}
&\mathsf{A}(S_{i,k}, 0, t, \mathcal{Q}) \\
&\qquad = e_i - \mathsf{A}(S_{i,k}, t, d_{i,k} + \Theta_i, \mathcal{Q}) \\
&\qquad \geq e_i - \min(e_i, d_{i,k} + \Theta_i - t) \\
&\qquad = \max(0, e_i - (d_{i,k} + \Theta_i - t)) \\
&\qquad = \max(0, e_i - (r_{i,k} + p_i + \Theta_i - t)) \\
&\qquad\quad \{\text{by (24)}\} \\
&\qquad = \max(0, e_i - \epsilon).
\end{aligned} \tag{26}$$

Since S_i's jobs are released periodically from time zero (since it is a server task), there are $\frac{r_{i,k}}{p_i}$ jobs released before job $S_{i,k}$. Thus,

$$\begin{aligned}
&\mathsf{A}(S_i, 0, t, \mathcal{Q}) \\
&\qquad = \mathsf{A}(S_{i,k}, 0, t, \mathcal{Q}) + \sum_{j<k} (\mathsf{A}(S_{i,j}, 0, t, \mathcal{Q})) \\
&\qquad\quad \{\text{by (25)}\} \\
&\qquad = \mathsf{A}(S_{i,k}, 0, t, \mathcal{Q}) + \frac{r_{i,k}}{p_i} \cdot e_i \\
&\qquad\quad \{\text{by (26)}\} \\
&\qquad \geq \max(0, e_i - \epsilon) + \frac{r_{i,k}}{p_i} \cdot e_i \\
&\qquad\quad \{\text{by (24)}\} \\
&\qquad = \max(0, e_i - \epsilon) + \frac{t - p_i - \Theta_i + \epsilon}{p_i} \cdot e_i \\
&\qquad = \max(0, e_i - \epsilon) + u_i \cdot t - u_i \cdot \Theta_i - e_i + \epsilon \cdot u_i \\
&\qquad = \max(u_i \cdot \epsilon - e_i, \epsilon \cdot (u_i - 1)) + u_i \cdot t - u_i \cdot \Theta_i
\end{aligned}$$

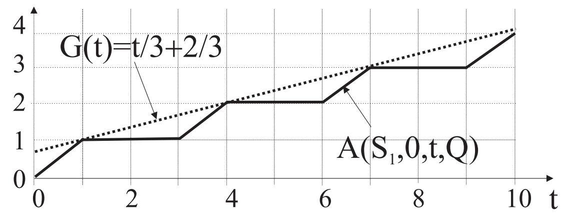


Figure 6: Server task allocation $\mathsf{A}(S_1, 0, t, \mathcal{Q})$ in Example 8 and its linear upper bound $G(t)$.

$$\begin{aligned}
&\left\{\begin{array}{l}\text{the } \max(\dots) \text{ summand is minimized if its two} \\ \text{arguments are equal, which is the case when } \epsilon = e_i\end{array}\right\} \\
&\geq \max(u_i \cdot e_i - e_i, e_i \cdot (u_i - 1)) + u_i \cdot t - u_i \cdot \Theta_i \\
&= u_i \cdot t - u_i \cdot \Theta_i - e_i \cdot (1 - u_i).
\end{aligned}$$
□

Example 8. Consider the schedule $\mathcal{Q}$ shown in Figure 3(b). In this schedule, jobs of the server task $S_1(1, 3)$ execute in the intervals $[0, 1)$, $[3, 4)$, and $[6, 7)$. By time 1, S_1 has received one allocation unit, by time 4, its allocation is two units, and so on. The allocation $\mathsf{A}(S_1, 0, t, \mathcal{Q})$ is shown in Figure 6 as a function of t. The figure also shows the upper bound (21), which is $G(t) \triangleq u_i \cdot t + e_i(1 - u_i) = 1/3 \cdot t + 1(1 - 1/3) = 1/3 \cdot t + 2/3$. It is easy to see that $\mathsf{A}(S_1, 0, t, \mathcal{Q}) \leq G(t)$.

We now can find guarantees on the supplied processor time for server tasks for an arbitrary time interval.

Theorem 2. *Suppose that the scheduling algorithm used by the container H ensures a deadline tardiness bound of Θ_i for the server task $S_i(e_i, p_i)$. Then S_i is guaranteed at least $\gamma_i^l(\Delta) = \max(0, u_i \cdot \Delta - 2e_i(1 - u_i) - u_i \cdot \Theta_i)$ time units during an interval of length Δ.*

Proof. Our goal is to bound the allocation of S_i during an interval $[t_1, t_2)$ by a function of the length of the interval $\Delta = t_2 - t_1$.

$$\begin{aligned}
&\mathsf{A}(S_i, t_1, t_2, \mathcal{Q}) \\
&\qquad = \mathsf{A}(S_i, 0, t_2, \mathcal{Q}) - \mathsf{A}(S_i, 0, t_1, \mathcal{Q}) \\
&\qquad\quad \{\text{by (21) and (22)}\} \\
&\qquad \geq u_i \cdot t_2 - u_i \cdot \Theta_i - e_i(1-u_i) - (u_i \cdot t_1 + e_i(1-u_i)) \\
&\qquad = u_i \cdot (t_2 - t_1) - 2e_i(1 - u_i) - u_i \cdot \Theta_i \\
&\qquad = u_i \cdot \Delta - 2e_i(1 - u_i) - u_i \cdot \Theta_i.
\end{aligned}$$

$\mathsf{A}(S_i, t_1, t_2, \mathcal{Q})$ cannot be less than zero, thus $\mathsf{A}(S_i, t_1, t_2, \mathcal{Q}) \geq \max(0, u_i \cdot \Delta - 2e_i(1 - u_i) - u_i \cdot \Theta_i)$. □

Corollary 3. *The supply to container C_i, as defined above, is described by $M(C_i) = \lceil w(C_i) \rceil$ availability functions in MP form, where $m(C_i) = \lfloor w(C_i) \rfloor$ supply functions satisfy $\beta_j^l(\Delta) = \Delta$ and at most one supply function satisfies $\beta_1^l(\Delta) = \gamma_i^l(\Delta)$ as given by Theorem 2. The total supplied bandwidth for C_i is $w(C_i)$.*

5.4 Computing Available Supply on HRT-Occupied Processors

In the previous section, we computed the supply available to a child container provided the tardiness bounds of tasks in τ_s are known. In order to calculate these tardiness bounds using Corollary 1, we need to determine the supply available to τ_s on $K(H)$ processors where HRT and SRT tasks are co-scheduled (if HRT capacity is reclaimed) in addition to the supply provided by the parent of H.

We first compute an upper bound on the allocation of an HRT task over the time interval $[t, t{+}\Delta)$.

Lemma 3. *If jobs of T_i finish by their deadlines in the schedule $\mathcal{Q}$, then $\mathsf{A}(T_i, t, t+\Delta, \mathcal{Q}) \leq u_i \cdot \Delta + 2 \cdot e_i \cdot (1 - u_i)$, for any t and $\Delta \geq 0$.*

Proof. Let $T_{i,k}$ be job of T_i with smallest index k that executes within $[t_1, t_2)$. If no such job exists, then T_i's allocation within $[t, t+\Delta)$ is zero and the required result holds trivially. Let $f_{i,k}$ be $T_{i,k}$'s completion time. The allocation of $T_{i,k}$ is thus

$$\mathsf{A}(T_{i,k}, t, t+\Delta, \mathcal{Q}) \leq \min(e_i, \Delta, \epsilon), \tag{27}$$

where $\epsilon = f_{i,k} - t$, as illustrated in Figure 7. We consider two cases based upon the relationship between ϵ and Δ.

Case 1: $\epsilon > \Delta$. In this case, $T_{i,k}$ commences execution at or before $t + \Delta$ and finishes after $t + \Delta$. By the selection of k, $T_{i,k}$ is the only job of T_i that executes within $[t, t + \Delta)$. Therefore, T_i's allocation in this interval cannot be greater than $\min(e_i, \Delta)$. By (27) and the condition of Case 1, we have

$$\begin{aligned}
&\mathsf{A}(T_{i,k}, t, t+\Delta, \mathcal{Q}) \\
&\qquad \leq \min(e_i, \Delta) \\
&\qquad = u_i \cdot \min(e_i, \Delta) + (1 - u_i) \cdot \min(e_i, \Delta) \\
&\qquad \leq u_i \cdot \Delta + (1 - u_i) \cdot e_i \\
&\qquad \leq u_i \cdot \Delta + 2 \cdot (1 - u_i) \cdot e_i.
\end{aligned}$$

Case 2: $\epsilon \leq \Delta$. Because, by the condition of the lemma, $T_{i,k}$ finishes by its deadline, $f_{i,k} \leq d_{i,k} = r_{i,k} + p_k \leq r_{i,k+1}$. The allocation of $T_{i,k}$'s successor jobs in the interval $[t, t + \Delta)$ is maximized if all of these jobs are released as soon as possible after $f_{i,k}$, as shown in Figure 7. Therefore,

$$\begin{aligned}
&\sum_{j>k} \mathsf{A}(T_{i,j}, t, t+\Delta, \mathcal{Q}) \\
&\qquad \leq \max\left(0, \left\lfloor \frac{t + \Delta - f_{i,k}}{p_i} \right\rfloor \cdot e_i + \min(e_i, (t + \Delta - f_{i,k}) \bmod p_i)\right) \\
&\qquad\quad \{\text{setting } f_{i,k} - t = \epsilon\} \\
&\qquad = \max\left(0, \left\lfloor \frac{\Delta - \epsilon}{p_i} \right\rfloor \cdot e_i + \min(e_i, (\Delta - \epsilon) \bmod p_i)\right).
\end{aligned} \tag{28}$$

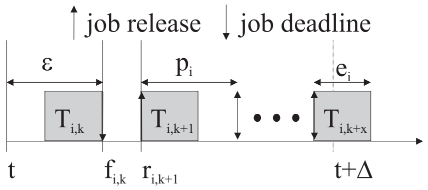


Figure 7: Maximum allocation scenario for a hard real-time task T_i.

By Definition 6 and the selection of k,

$$
\begin{aligned}
&\mathsf{A}(T_i, t, t+\Delta, \mathcal{Q}) \\
&\quad = \mathsf{A}(T_{i,k}, t, t+\Delta, \mathcal{Q}) + \sum_{j>k} \mathsf{A}(T_{i,j}, t, t+\Delta, \mathcal{Q}) \\
&\qquad \{\text{by (27) and (28)}\} \\
&\quad \le\ \min(e_i, \Delta, \epsilon) + \max\left(0, \left\lfloor \frac{\Delta-\epsilon}{p_i} \right\rfloor \cdot e_i + \min(e_i, (\Delta-\epsilon)\ mod\ p_i)\right) \\
&\qquad \{\text{by the condition of Case 2}\} \\
&\quad =\ \min(e_i, \epsilon) + \left\lfloor \frac{\Delta-\epsilon}{p_i} \right\rfloor \cdot e_i + \min(e_i, (\Delta-\epsilon)\ mod\ p_i) \\
&\qquad \left\{\text{setting } \frac{\Delta-\epsilon}{p_i} = q\right\} \\
&\quad =\ \min(e_i, \epsilon) + \lfloor q \rfloor \cdot e_i + \min(e_i, q \cdot p_i - \lfloor q \rfloor \cdot p_i) \\
&\quad =\ \min(e_i, \epsilon) + q \cdot e_i - q \cdot e_i + \lfloor q \rfloor \cdot e_i + \min(e_i, q \cdot p_i - \lfloor q \rfloor \cdot p_i) \\
&\quad =\ \min(e_i, \epsilon) + q \cdot e_i + \min(e_i \cdot (\lfloor q \rfloor - q + 1), (q - \lfloor q \rfloor) \cdot (p_i - e_i)) \\
&\qquad \{\text{setting } q - \lfloor q \rfloor = z\} \\
&\quad =\ \min(e_i, \epsilon) + q \cdot e_i + \min(e_i \cdot (1-z), z \cdot (p_i - e_i)) \\
&\qquad \left\{ \begin{array}{l} \min(e_i \cdot (1-z), z \cdot (p_i - e_i)) \text{ is maximized if both its} \\ \text{arguments are equal, which is the case when } z = u_i \end{array} \right\} \\
&\quad \le\ \min(e_i, \epsilon) + q \cdot e_i + \min(e_i \cdot (1-u_i), u_i \cdot (p_i - e_i)) \\
&\qquad \left\{\text{setting } q = \frac{\Delta-\epsilon}{p_i}\right\} \\
&\quad =\ \min(e_i, \epsilon) + \frac{\Delta-\epsilon}{p_i} \cdot e_i + e_i \cdot (1-u_i) \\
&\quad =\ \min(e_i, \epsilon) + (\Delta-\epsilon) \cdot u_i + e_i \cdot (1-u_i) \\
&\qquad \{\text{maximized if } \epsilon = e_i\} \\
&\quad \le e_i + (\Delta - e_i) \cdot u_i + e_i \cdot (1-u_i) \\
&\quad = u_i \cdot \Delta + 2 \cdot e_i \cdot (1-u_i).
\end{aligned}
$$

□

Lemma 4. *Let τ_h be the set of HRT tasks assigned to a fully available processor h such that $U_{sum}(\tau_h) < 1$. For any time interval of length Δ, at least $\beta_h^l(\Delta) = \max(0, \widehat{u_h} \cdot (\Delta - \sigma_h))$ time units are available, where $\widehat{u_h} = 1 - U_{sum}(\tau_h)$ and $\sigma_h = \frac{2\sum_{T_i \in \tau_h} e_i \cdot (1-u_i)}{1-U_{sum}(\tau_h)}$.*

Proof. Consider an interval $[t, t+\Delta)$. By Definition 6, the time available after scheduling τ_h within this interval is

$$
\begin{aligned}
&\max\left(0, \Delta - \sum_{T_i \in \tau_h} \mathsf{A}(T_i, t, t+\Delta)\right) \\
&\quad \{\text{setting } t_1 = t \text{ and } t_2 = t+\Delta \text{ into Lemma 3}\} \\
&\geq \max\left(0, \Delta - \sum_{T_i \in \tau_h} (u_i \cdot \Delta + 2 \cdot e_i \cdot (1-u_i))\right) \\
&\quad \{\text{by (1)}\} \\
&\geq \max\left(0, \Delta \cdot (1 - U_{sum}(\tau_h)) - 2 \cdot \sum_{T_i \in \tau_h} e_i \cdot (1-u_i)\right) \\
&\quad \{\text{by the definition of } \widehat{u_h} \text{ and } \sigma_h \text{ in the statement of the lemma}\} \\
&= \max(0, \widehat{u_h} \cdot (\Delta - \sigma_h)).
\end{aligned}
$$

□

Definition 7. Let $M(H)$ be the total number of processors that provide supply to H. Let $Y = M(H) - \sum_{C_j \in succ(H)} m(C_j) - m(C_{hrt})$ be the number of processors that are not reserved for HRT tasks and child containers of H.

The following theorem summarizes the analysis discussed in the previous sections. In the statement of the theorem, G, $K(H)$, and τ_s are as defined earlier in Definitions 1 and 2, and $M_s = Y + K(H)$.

Theorem 3. *If the host container H's supply is in MP form, then hard real-time schedulability for HRT tasks and bounded deadline tardiness for SRT and server tasks encapsulated in H are guaranteed if (6) holds (with M is replaced with M_s and τ is replaced with τ_s) and (14) holds. If deadline tardiness is bounded for a server task, then the supply to the corresponding child container is in MP form and the supplied bandwidth matches that specified for the child container.*

Proof. We illustrate the proof using Figure 8. In this figure, the supply available to H is represented as $M(H)$ bins for which the height of the bin represents the available utilization on the respective processor. We first dedicate an integral number of processors to supply the integral part of the child containers' bandwidths (these processors are shaded black). We then partition the tasks in $\mathsf{HRT}(H)$ among $m(C_{hrt})$ processors and find the number Y as defined in Definition 7. For each processor h such that $h \in [Y+1, Y+m(C_{hrt})]$, we find the unused bandwidth $\widehat{u_h} = 1 - U_{sum}(\tau_h)$ using Lemma 4, as shown in Figure 8. After determining $K(H)$, we find $M_s = Y + K(H)$ and the bandwidth available to SRT and server tasks (this bandwidth is shaded light gray in Figure 8). In order to apply Corollary 1, we need to re-number the processors with indices 1 to M_s so that partially available processors are listed first. Finally, we apply Corollary 1 to calculate tardiness bounds for the tasks in τ_s and use Corollary 3 to find the supply functions for child containers. Each child container $C_j \in succ(H)$ is thus guaranteed supply from an integral number of fully available

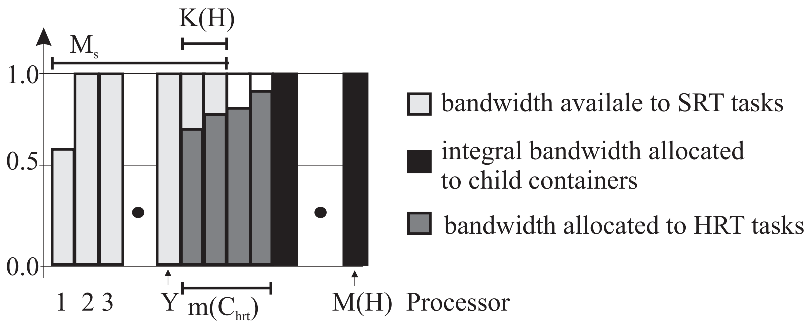


Figure 8: Illustration of Theorem 3.

processors plus the time allocated on an additional processor whenever the respective server task S_j is scheduled. By Corollary 3, this allocation is proportional to S_j's utilization, which is the fractional part of C_j's bandwidth. Therefore, the supplied bandwidth to each child container C_j is proportional to its required bandwidth and is in MP form. □

Applying the above theorem recursively, we can analyze the properties of a container hierarchy. Note that the tardiness of SRT tasks may be higher as compared to a corresponding non-hierarchical approach, where all tasks are scheduled at the same level because the degree of parallelism of the available supply is lower under our approach. This is the price for having temporal isolation among containers. In Section 7, we discuss in greater detail the conditions under which temporal isolation is guaranteed.

6 Tradeoffs for HRT Tasks

If there are no HRT tasks in the system, then no utilization loss is incurred. If the system has HRT tasks, then tradeoffs between the schedulability and tardiness of SRT tasks and utilization loss are possible, as illustrated by the example below.

Example 9. Consider a container H encapsulating three HRT tasks T_1, T_2, and T_3 with utilization 0.51 and six SRT tasks $\mathsf{SRT}(H) = \{T_4, \ldots, T_9\}$ with utilization 0.5 as shown in Figure 9(a). H's bandwidth of $w(H) = 5.5$ is supplied by a partially available processor 1 with $\widehat{u_1} = 0.5$ and five fully available processors, as shown in Figure 9(b). In this figure, the processors are represented as six bins. By (3), the HRT tasks require three dedicated processors since no two of these tasks can be assigned to one processor without violating HRT constraints. These tasks are therefore assigned to processors 4–6. The bandwidth consumed by the HRT tasks is shaded. After the HRT tasks are allocated, the total bandwidth provided by processors 1–3, which is 2.5, is insufficient to handle all SRT tasks, whose total utilization is $U_{sum}(\mathsf{SRT}(H)) = 3$. We reclaim the unused bandwidth on processors 4 and 5 by setting $K(H) = 2$ (see Definition 1). The supply available to the SRT tasks is now given by $M_s = 5$ processors with utilizations $\widehat{u_1} = 0.49$, $\widehat{u_2} = 0.49$, $\widehat{u_3} = 0.5$, $\widehat{u_4} = 1.0$, and $\widehat{u_5} = 1.0$, respectively. (Note that processors are ordered by increasing utilizations. The first two utilization values were obtained using Lemma 4.) The total supplied bandwidth is thus $\sum_{k=1}^{M_s} \widehat{u_k} = 3.48$, which exceeds the total utilization of the SRT tasks, and hence, (6) holds.

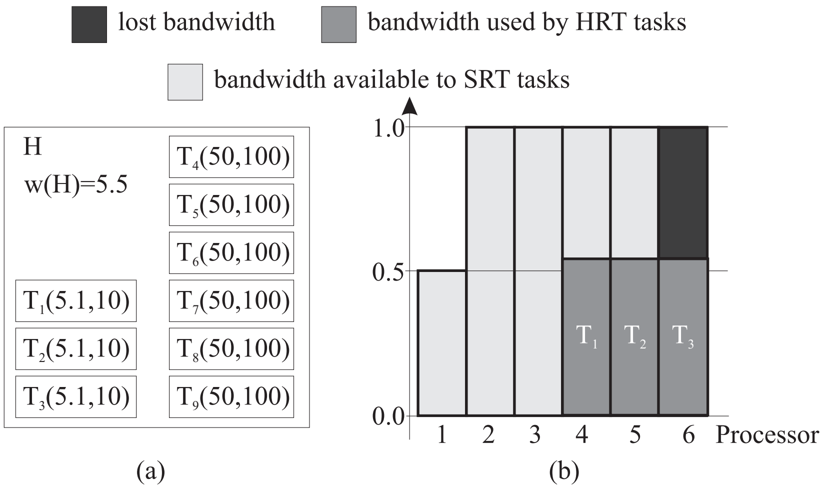


Figure 9: Bandwidth allocation and utilization loss in Example 9.

Because the supply to the SRT tasks is not in MP form (i.e., more than one processor is partially available), by Corollary 1, we have to test whether (14) holds in order to check the schedulability of $T_4, \ldots, T_9$. Setting the supply and task parameters into (14), we have

$$\begin{aligned} M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} - \max(K(H) + G - 1, 0) \max_{1 \le \ell \le |\tau_s|} (u_\ell) - U_L \\ \left\{ \begin{array}{l} \text{because } M_s = 5, G = 1, K(H) = 2, \max(u_\ell) = 0.5, \\ \text{and } U_L = (M_s - 1) \cdot 0.5 = 2 \end{array} \right\} \\ = 5 - 2 - 1 + (0.49 + 0.49 + 0.5) - \max(3 - 1, 0) \cdot 0.5 - 2 \\ = 0.48 \\ > 0. \end{aligned}$$

Thus, bounded tardiness for the SRT tasks is guaranteed if $K(H) = 2$. Also, the utilization loss, which is the bandwidth that is unused by HRT tasks and that is unavailable to SRT tasks, is 0.49 in this case (this unused utilization is shaded black in Figure 9(b)). If we try to reduce the utilization loss even further by setting $K(H)$ to 3, then, even though the total utilization available to the SRT tasks becomes 3.97, (14) no longer holds.

The example above shows that the co-scheduling of HRT and SRT tasks may be necessary in order to accommodate a workload using the supplied bandwidth. However, SRT schedulability can be compromised for large $K(H)$ due to (14). To find the maximum $K(H)$ so that the tasks in τ_s remain schedulable, we can apply Theorem 3 for each $K(H)$ from $m(C_{hrt})$ to zero.

From (14) and (9), we conclude that (14) is more likely to hold if $K(H)$ or $\max_{1 \le \ell \le |\tau_s|}(u_\ell)$ is small. Therefore, reclaiming processor time can be successful if the maximum per-task utilization of SRT and server tasks is small.

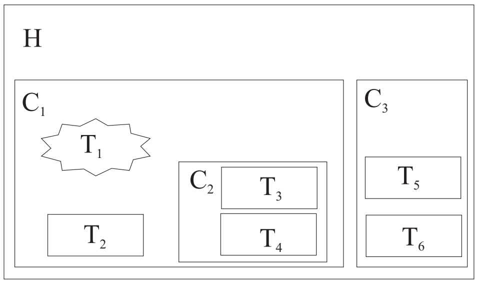


Figure 10: Container isolation.

7 Misbehaving Tasks

We call a task T_i *misbehaving* if its worst-case execution time may exceed e_i. In this section, we describe the impact of misbehaving tasks on a system and show how to alleviate any adverse effects. Consider the container configuration shown in Figure 10. In this figure, T_1 is a misbehaving task and is denoted by a star-shaped outline. In the configuration shown in Figure 10, the processor supplies of C_1 and C_3 depend solely on the supply of H and the parameters of the server tasks S_1 and S_3, which cannot be misbehaving since a server task is not scheduled when its budget is depleted. By Corollary 3, the parameters of S_3 and its deadline tardiness define the guaranteed supply of C_3, and hence, the tardiness of T_5 and T_6. Thus, the misbehaving task T_1 does not affect the timeliness of tasks belonging to C_3. That is, the tasks in container C_3 are *temporally isolated* from the misbehaving task. More generally, any two tasks $T_i \in succ(C_k)$ and $T_j \in succ(C_l)$ are temporally isolated iff C_k is not a member of the hierarchy rooted at C_l and C_l is not a member of the hierarchy rooted at C_k.

On the other hand, a misbehaving task T_i *can* affect the timeliness of tasks encapsulated in that part of container hierarchy that is rooted at T_i's parent. In our example, due to the misbehaving task T_1, task T_2's tardiness may exceed its computed bound. As a consequence, the tardiness of the server task S_2 of container C_2 may exceed its computed bound thereby invalidating the bounds on processor allocation for container C_2. This, in turn, may affect the timeliness of the encapsulated tasks T_3 and T_4. To prevent such problems, any potentially misbehaving task should be isolated in a container for which a budget can be enforced.

8 Experiments

We now present the results of experiments conducted to compare our container-aware scheduling scheme with conventional scheduling techniques. In these experiments, performance was compared using randomly-generated task sets, which have both HRT and SRT tasks.

Task generation procedure. In order to gain intuition about the properties of a large multiprocessor platform running multiple isolated components, we evaluated a three-level container hierarchy consisting of a root container C_0, four second-level containers, and then the contained tasks, as shown in Figure 11. The i-th second-level container is denoted $C_{sys}^{[i]}$ and its contained HRT and SRT tasks as $\tau_{hrt}^{[i]}$ and $\tau_{srt}^{[i]}$, respectively. Randomly-generated tasks were added to these sets while

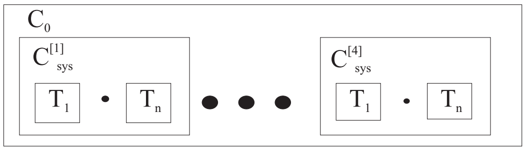


Figure 11: Experimental setup.

$U(\tau_{hrt}^{[i]})$ is at most $U_{hrt} \leq 1$ and $U(\tau_{srt}^{[i]}) \leq 3.5$. Task utilizations were taken randomly from $[0, 0.15)$ for HRT tasks and from $[u_{min}, u_{max})$ for SRT tasks. We examined three HRT total utilization caps U_{hrt} and four SRT utilization ranges, as described later. Integral task periods were taken randomly from $[100, 1000]$ for HRT tasks and from $[10000, 50000]$ for SRT tasks. Integral execution times were computed using periods and utilizations.

We compared our container-aware scheduling scheme (CA) with PEDF and a hybrid EDF-based scheme (HS), both of which are oblivious to containers. The HS scheme, which is described later in this section, is a naïve combination of PEDF and GEDF. PEDF was selected because it exhibits good timeliness, and HS was selected because it can satisfy the requirements of HRT and SRT tasks using relatively few processors. However, HS and PEDF do not provide any isolation among containers. In our experiments, we compared the tested schemes based on *the required number of processors* (RNP) and *deadline tardiness bound* (TB). We did not consider any system overheads or other container hierarchies. Such things are very application- and implementation-specific, respectively, and our intent here is only to provide a basic sense of how our scheme compares to the other implementation alternatives.

Defining RNP. Under PEDF, RNP is defined as the minimum number of processors required to partition all real-time tasks using the first-fit heuristic. Under PEDF, all tasks have zero tardiness.

Under HS, HRT and SRT tasks run on disjoint processor sets, with all HRT tasks scheduled together using PEDF with the first-fit heuristic, and all SRT tasks scheduled together using GEDF. RNP for the SRT tasks is thus

$$M_{soft} = \left\lceil \sum_{i=1}^{4} U_{sum}(\tau_{srt}^{[i]}) \right\rceil .$$

Letting M_{hard} denote the HRT RNP, overall RNP under HS is simply $M_{hard} + M_{soft}$.

Under CA, we set container $C_{sys}^{[i]}$'s bandwidth to $w(C_{sys}^{[i]}) = W_I + W_f$ where W_I is the number of required fully available processors, and W_f is the minimum utilization due to (at most one) partially available processor. As explained next, W_I and W_f were determined based upon whether it is possible to reclaim bandwidth not used by HRT tasks (we illustrate this explanation with an example below). Because $U(\tau_{hrt}^{[i]}) \leq U_{hrt} \leq 1$, the HRT tasks of each second-level container require at most one processor. We checked whether any bandwidth on this processor can be reclaimed for SRT tasks as follows. We set $K_r(H) = 1$ (reclaiming is possible) if $\tau_{srt}^{[i]}$ is schedulable on $\left\lceil U_{sum}(\tau_{hrt}^{[i]} \cup \tau_{srt}^{[i]}) \right\rceil$ processors such that one processor has an available utilization of $1 - U_{sum}(\tau_{hrt}^{[i]})$ and one processor has an available utilization of $\mathsf{frac}(U_{sum}(\tau_{hrt}^{[i]} \cup \tau_{srt}^{[i]}))$, where $\mathsf{frac}(x)$ is the fractional part of x. Otherwise, we set $K_r(H) = 0$ (i.e., reclaiming is not possible).

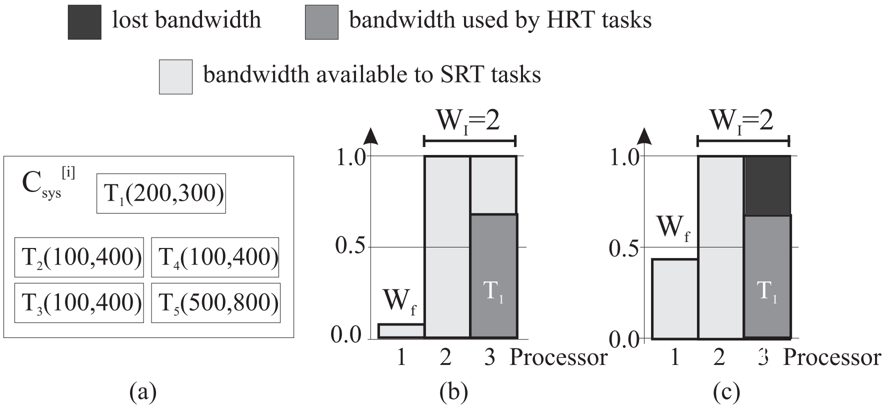


Figure 12: Determining the required container bandwidth in Example 10.

After the degree of reclamation was determined, we set

$$W_I = \begin{cases} \lfloor U_{sum}(\tau_{srt}^{[i]} \cup \tau_{hrt}^{[i]}) \rfloor & \text{if } K_r(H) = 1 \\ \lfloor U_{sum}(\tau_{srt}^{[i]}) \rfloor + 1 & \text{otherwise.} \end{cases}$$

The fractional part of the bandwidth W_f was set to

$$W_f = \begin{cases} \mathsf{frac}(U_{sum}(\tau_{srt}^{[i]} \cup \tau_{hrt}^{[i]})) & \text{if } K_r(H) = 1 \\ \mathsf{frac}(U_{sum}(\tau_{srt}^{[i]})) & \text{otherwise.} \end{cases}$$

Example 10. Consider container $C_{sys}^{[1]}$ with HRT task $T_1(200, 300)$ and SRT tasks $T_2(100, 400), \ldots,$ $T_4(100, 400)$, and $T_5(500, 800)$ as shown in Figure 12(a). (Note that these task parameters are not allowed by out task generation method; however, allowing them simplifies the example.) For this task set, $U_{sum}(\tau_{hrt}^{[1]}) = 2/3$, $U_{sum}(\tau_{srt}^{[1]}) = 11/8$, and $U_{sum}(\tau_{srt}^{[1]} \cup \tau_{hrt}^{[1]}) = 49/24$. We first check the schedulability of $T_2, \ldots, T_5$ on $\left\lceil U_{sum}(\tau_{srt}^{[1]} \cup \tau_{hrt}^{[1]}) \right\rceil = 3$ processors such that one processor is fully available and two processors have available utilizations of $1/24$ and $1/3$ (see processors 1 and 3 in Figure 12(b)). It can be shown that (14) does does not hold for this task system, and hence, we have to set $K(H) = 0$. With this setting of $K(H)$, we cannot co-schedule the HRT and the SRT tasks on processor 3. It can be verified that the SRT tasks are schedulable on $\lceil U_{sum}(\tau_{srt}^{[1]}) \rceil = 2$ processors such that one processor is fully available and one processor has an available utilization of $\mathsf{frac}(U_{sum}(\tau_{srt}^{[1]})) = 3/8$ (see processors 1 and 2 in Figure 12(b)). Therefore, we set $W_I = 2$, since the HRT and the SRT tasks together require two fully available processors, and $W_f = 3/8$, because the SRT tasks additionally need a bandwidth of $\mathsf{frac}(U_{sum}(\tau_{srt}^{[1]})) = 3/8$.

The execution time e_i and the period p_i of the server task $S_{srt}^{[i]}$ should be set such that $e_i/p_i = W_f$. Once W_f has been determined and a value for e_i is selected, p_i is implicitly determined. However, a tradeoff exists in selecting e_i. On one hand, a smaller value of e_i effectively reduces the server task's maximum tardiness, and correspondingly, the supply blackout time, as (13) and

Theorem 2 suggest. On the other hand, small server task execution times could lead to frequent context switches in a real implementation. As a compromise, we set the execution time of each server task to be 100, which is close to the average execution time of SRT tasks in H. Server tasks' periods were set to $\left\lfloor \frac{100}{W_f} \right\rfloor$ if $W_f \neq 0$, so that the utilization of the server task is slightly higher than the fractional part of the required container bandwidth. The required container bandwidth $C_{sys}^{[i]}$ was then inflated accordingly by

$$u_S = \begin{cases} 0 & \text{if } W_f = 0 \\ \frac{100}{\lfloor 100/W_f \rfloor} - W_f & \text{otherwise,} \end{cases}$$

where u_S is the utilization loss associated with the choice of server task parameters.

As an example consider container $C_{sys}^{[i]}$ from Example 10. Because $W_f = 3/8$, we set $u_S = \frac{100}{\lfloor 100/(3/8) \rfloor} - 3/8 = 0.001$.

Overall, RNP for CA is simply the bandwidth of the root container C_0, $w(C_0) = \lceil \sum_{i=1}^{4} w(C_{sys}^{[i]}) \rceil$.

RNP **results.** Insets (a), (c), (e), and (g) of Figure 13 show RNP results for PEDF, HS, and CA, for the SRT utilization ranges $[0.01, 0.1)$ (light), $[0.1, 0.5)$ (medium), and $[0.5, 1)$ (heavy), respectively. We also examined the SRT utilization range $[0.5,\ 0.7)$ (extreme) as well, as it is an extreme case where PEDF shows poor performance. The x axis in each inset corresponds to the HRT utilization cap, U_{hrt}.

For each utilization range, 100 task sets were generated and their RNP averaged. The figure also shows the average total system utilization, so that we can estimate the utilization loss associated with each scheme.

For the light and medium SRT per-task utilization ranges (insets (a) and (c)), all three schemes show similar performance. This is because CA is able to minimize the bandwidth of individual second-level containers by co-scheduling HRT and SRT tasks together. As SRT per-task utilization increases, RNP for PEDF also increases because more processors are needed to bin-pack the SRT tasks. The extreme case (inset (g)) is the utilization range $[0.5, 0.7)$, where each SRT task requires a separate processor.

When SRT per-task utilizations are large (inset (e)), the difference between HS and CA is maximal, due to the utilization loss associated with HRT tasks in the containers. Under CA, the four HRT task sets require four processors, while under HS, all HRT tasks may be packed onto a smaller number of processors.

Tardiness. Insets (b), (d), (f), and (h) of Figure 13 show the average of the per-task-set tardiness bounds under HS and CA for the task set categories discussed above (under PEDF, tardiness is zero). For these two schemes, these tardiness bounds are comparable in most cases, with the tardiness under CA being slightly higher due to uneven supply by the server tasks. Under CA, the maximum tardiness bound is significantly higher when the maximum total utilization of HRT tasks is high (see the HRT utilization cap of 0.9 in insets (b) and (d) of Figure 13). This is because CA attempts to reclaim scarce processor supply available after scheduling HRT tasks within the container and use that supply to schedule SRT tasks. However, even though the maximum task tardiness in these cases is higher, the number of processors required by CA is lower (see insets (a) and (c) of Figure 13).

Overall, these experiments show that in some cases there is a price to be paid for temporal isolation among containers, in the form of more required processors (if HRT tasks are present) or higher tardiness. However, in our proposed scheme, this price is reasonable, when considering the performance of schemes that ensure no isolation.

As a final comment, we remind the reader that if no HRT tasks are present, then our scheme incurs *no* utilization loss.

9 Conclusion

We have presented a multiprocessor bandwidth-reservation scheme for hierarchically organized real-time containers. Under this scheme each real-time container can reserve any fraction of processor time (even the capacity of several processors) to schedule its children. The presented scheme provides temporal isolation among containers so that each container can be analyzed separately.

Our scheme is novel in that soft real-time components incur no utilization loss. This stands in sharp contrast to hierarchical schemes for hard (only) real-time systems, where the loss per level can be so significant, arbitrarily deep hierarchies simply become untenable.

Several interesting avenues for further work exist. The most important open problem is to enable dynamic container creation and the joining/leaving of tasks. Also of importance is the inclusion of support for synchronization. It would also be interesting to investigate other global scheduling algorithms such as Pfair algorithms to see whether a more accurate analysis can be established for them. Finally, the new scheduling policy needs to be implemented on a real multiprocessor platform so that overheads associated with the hierarchical nature of the system could be determined. Given that work on Linux containers partially motivated our research, a Linux-based system such as $\mathrm{LITMUS}^{\mathrm{RT}}$ ($\mathrm{LITMUS}^{\mathrm{RT}}$ homepage, 2008), where GEDF is implemented along with other global scheduling algorithms, would be desirable to use in such an effort.

Acknowledgement: Work supported by grants from Intel and IBM Corps., by NSF grants CNS 0408996, CCF 0541056, and CNS 0615197 and by ARO grant W911NF-06-1-0425.

References

(2007) Linux vserver documentation Http://linux-vserver.org/Documentation

Anderson J, Calandrino J, Devi U (2006) Real-time scheduling on multicore platforms. In: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, pp 179–190

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

Baruah S, Baker T (2008) Global EDF schedulability analysis of arbitrary sporadic task systems. In: Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp 3–12

Baruah S, Fisher N (2005) The partitioned multiprocessor scheduling of sporadic task systems. In: Proceedings of the 26th IEEE Real-Time Systems Symposium, pp 321–329

Baruah S, Lipari G (2004) A multiprocessor implementation of the total bandwidth server. In: Proceedings of 18th International Parallel and Distributed Processing Symposium, p 40

Baruah S, Goossens J, Lipari G (2002) Implementing constant-bandwidth servers upon multiprocessor platforms. In: Proceedings of the IEEE International Real-Time and Embedded Technology and Applications Symposium, pp 154–163

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

Brandenburg B, Anderson J (2007) Integrating hard/soft real-time tasks and best-effort jobs on multiprocessors. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems, pp 61–70

Chakraborty S, Thiele L (2005) A new task model for streaming applications and its schedulability analysis. In: Proceedings of the IEEE Design Automation and Test in Europe (DATE), pp 486–491

Devi U, Anderson J (2008) Tardiness bounds for global EDF scheduling on a multiprocessor. Real-Time Systems 38(2):133–189

Easwaran A, Anand M, Lee I (2007) Compositional analysis framework using EDP resource models. In: Proceedings of the 28th IEEE Real-Time Systems Symposium, pp 129–138

Eriksson M, Palmroos S (2007) Comparative study of containment strategies in solaris and security enhanced Linux Http://opensolaris.org/os/community/security/news/20070601-thesis-bs-eriksson-palmroos.pdf

Holman P, Anderson J (2006) Group-based Pfair scheduling. Real-Time Systems 32(1-2):125–168

LITMUS$^{\mathrm{RT}}$ homepage (2008) Http://www.cs.unc.edu/~anderson/litmus-rt

Leontyev H, Anderson J (2008a) Generalized tardiness bounds for global multiprocessor scheduling. Real-time Systems In submission, Preliminary version appeared in proceedings of the 28th Real-Time Systems Symposium, pp 413–422, 2007

Leontyev H, Anderson J (2008b) A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In: Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp 191–200

Lessard P (2003) Linux process containment: A practical look at chroot and user mode Linux Http://www.sans.org/reading_room/whitepapers/linux/1073.php

Liu C, Layland J (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM 30:46–61

Liu J (2000) Real-Time Systems. Prentice Hall

Moir M, Ramamurthy S (1999) Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In: Proceedings of the 20th IEEE Real-Time Systems Symposium, pp 294–303

Pellizzoni R, Caccamo M (2008) M-CASH: A real-time resource reclaiming algorithm for multiprocessor platforms. Real-Time Systems 40(1):117–147

Rajkumar R (2006) Resource Kernels: Why Resource Reservation should be the Preferred Paradigm of Construction of Embedded Real-Time Systems. Keynote talk, 18th Euromicro Conference on Real-Time Systems, Dresden, Germany

Shin I, Easwaran A, Lee I (2008) Hierarchical scheduling framework for virtual clustering of multiprocessors. In: Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp 181–190

Srinivasan A, Holman P, Anderson J (2002) Integrating aperiodic and recurrent tasks on fair-scheduled multiprocessors. In: Proceedings of the 14th Euromicro Conference on Real-Time Systems, pp 19–28

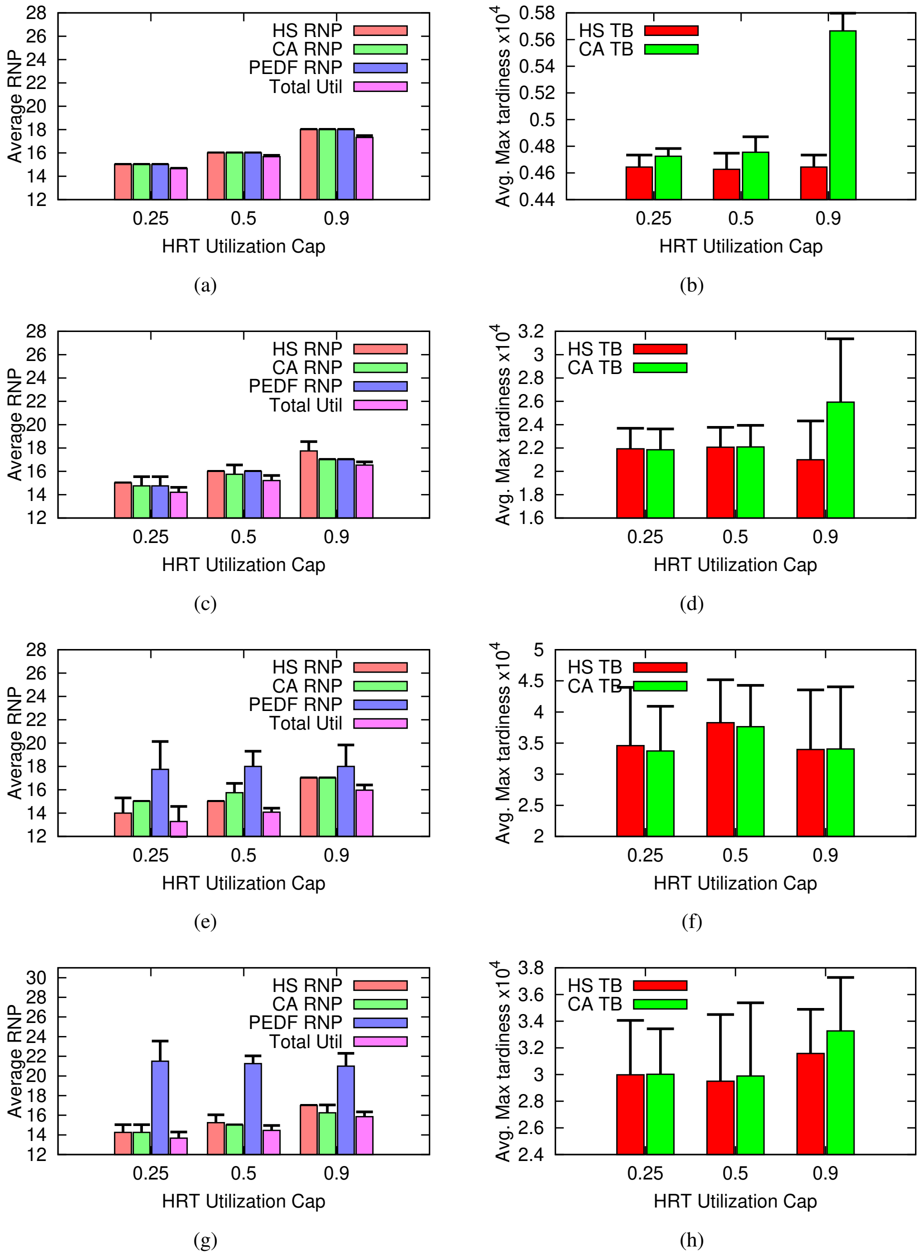


Figure 13: **(a,c,e,g)** Required number of processors and **(b,d,f,h)** maximum tardiness bounds for randomly generated task sets (with 95% confidence intervals) for **(a)**–**(b)** light, **(c)**–**(d)** medium, **(e)**–**(f)** heavy, and **(g)**–**(h)** extreme SRT utilization distributions.