

## **Response Time versus Utilization in Scheduler Overhead Accounting**

Silviu S. Craciunas      Christoph M. Kirsch      Ana Sokolova

Technical Report 2009-03      September 2009

**Department of Computer Sciences**

Jakob-Haringer-Straße 2  
5020 Salzburg  
Austria  
[www.cosy.sbg.ac.at](http://www.cosy.sbg.ac.at)

**Technical Report Series**

# Response Time versus Utilization in Scheduler Overhead Accounting\*

Silviu S. Craciunas      Christoph M. Kirsch      Ana Sokolova

Department of Computer Sciences  
University of Salzburg, Austria  
`firstname.lastname@cs.uni-salzburg.at`

**Abstract.** We propose two complementary methods to account for scheduler overhead in the schedulability analysis of Variable Bandwidth Servers (VBS), which control process execution speed by allocating variable CPU bandwidth to processes. Scheduler overhead in VBS may be accounted for either by decreasing process execution speed to maintain CPU utilization (called response accounting), or by increasing CPU utilization to maintain process execution speed (called utilization accounting). Both methods can be combined by handling an arbitrary fraction of the total scheduler overhead with one method and the rest with the other. Distinguishing scheduler overhead due to releasing and due to suspending processes allows us to further improve our analysis by accounting for releasing overhead in a separate, virtual VBS process, which increases CPU utilization less than through regular utilization accounting. The remaining overhead may then be accounted for in either more increased CPU utilization, or decreased process execution speed. Our method can readily be generalized to account for non-constant scheduler overhead, motivated by our VBS implementation, which can be configured to run (with increasing space complexity) in quadratic, linear, and constant time (in the number of processes). Although our analysis is based on the VBS model, the general idea of response and utilization accounting may also be applied to other, related scheduling methods.

## 1 Introduction

We study scheduler overhead accounting in the schedulability analysis of Variable Bandwidth Servers (VBS) [10, 12]. VBS are a generalization of Constant Bandwidth Servers (CBS) [1]. A CBS allocates a constant fraction of CPU time to a process (the server bandwidth) at a constant granularity (the server period). Multiple CBS processes are EDF-scheduled using the server periods as deadlines. A VBS is similar to a CBS but also allows the process to change its execution speed at any time, i.e., change both server bandwidth and server period, as long as the resulting CPU utilization remains under a given bandwidth cap. The portion of process code from a change in speed to the next is called an action. A

---

\* Supported by the EU ArtistDesign Network of Excellence on Embedded Systems Design and the Austrian Science Funds P18913-N15 and V00125.

VBS process is therefore a sequence of actions. The total time to execute an action is explicitly modeled and called the response time of the action. The key result of VBS is that, for each action of a VBS process, there exist lower and upper bounds on response times and thus also on jitter that are independent of any other concurrently executing VBS processes, as long as system utilization (the sum of all bandwidth caps) is less than or equal to 100% [10, 12] (Section 3).

This paper generalizes the result to include the overhead of scheduler execution. We first determine an upper bound on the number of scheduler invocations that may occur during a given amount of time (Section 4). This is possible because process release and suspend times are known in VBS. We then show that there are two complementary methods to account for scheduler overhead, either by decreasing the speed at which processes run to maintain CPU utilization (called response accounting), or by increasing CPU utilization to maintain the speed at which processes run (called utilization accounting). Response accounting decreases the net server bandwidth available to a process by dedicating some of its bandwidth to the scheduler. Utilization accounting increases the server bandwidth to maintain the net bandwidth available to a process. In other words, with utilization accounting, the bounds on response times are maintained while CPU utilization is increased whereas, with response accounting, the upper bounds on response times and thus on jitter are increased while utilization is maintained. Both methods can be combined by handling an arbitrary fraction of the total scheduler overhead with one method and the rest with the other. We also show, by example, that the fraction may be chosen within server- and process-dependent intervals such that CPU utilization decreases while the response-time bounds and thus the speed at which processes run remain the same (Section 5).

Next, we observe that there is a natural division of the VBS scheduler overhead into overhead due to releasing and due to suspending processes. Moreover, we note that our previously mentioned upper bound on the number of scheduler invocations can be improved for the scheduler invocations due to releasing processes by accounting for them in a separate, virtual VBS process instead of the given VBS processes (Section 6). The virtual process is then accounted for in increased CPU utilization yet less than in regular utilization accounting. The remaining overhead due to suspending processes may then be accounted for in either more increased CPU utilization, or increased response-time bounds and thus decreased speed at which processes run. The former method is pure utilization accounting, the latter method is combined accounting. Pure response accounting does not apply here.

Up until this point we have assumed that there is a constant upper bound on the execution time of any scheduler invocation. The VBS scheduling algorithm itself is indeed constant-time. Even queue management in VBS can be done in constant time. However, our VBS implementation features a plugin architecture for queue management so that other plugins with non-constant-time complexity (but less space complexity) can be used. In addition to the constant-time plugin, there are also a list-based, quadratic-time and an array-based, linear-time plugin

(in the number of processes) [10, 12]. We briefly show that our method can readily be generalized to account for non-constant scheduler overhead (Section 7) and then conclude the paper (Section 8).

Note that, although our analysis is based on the VBS model, the general idea of response and utilization accounting may also be applied to CBS as well as to RBED [4], which is another rate-based scheduler closely related to VBS. More detailed related work is discussed next.

## 2 Related Work

We first put Variable Bandwidth Servers (VBS) [10, 12] in the context of earlier work on scheduling. We then identify examples of response and utilization accounting in related work on scheduler overhead. We also acknowledge previous work on a wider range of issues dealing with general system overhead.

The Generalized Processor Sharing (GPS) approach introduced an idealized model of resources for fair scheduling based on the assumption that resource capacities are infinitely divisible [21]. Proportional-share algorithms (PSA) such as [13] approximate GPS using quantum-based CPU scheduling techniques. Constant Bandwidth Servers (CBS) [1] are related to PSA [2] but allocate a constant fraction of CPU time to each process (server bandwidth) at a constant granularity (server period). VBS is a generalization of CBS that allows processes to change their servers' bandwidth and period (also called virtual periodic resource, or resource reservation [22]) at runtime as long as the CPU capacity is not exceeded.

VBS is closely related to RBED, which is a rate-based scheduler extending resource reservations for hard real-time, soft real-time, and best effort processes [4]. Like VBS, RBED uses EDF scheduling and allows dynamic bandwidth and rate adjustments. While the capabilities are similar, RBED and VBS differ on the level of abstractions provided. In VBS we model processes as sequences of actions to quantify the response times of portions of process code, where each transition from one action to the next marks an adjustment in bandwidth and rate.

Other scheduling techniques related to VBS include elastic scheduling [8], handling of quality of service [20] and overload scenarios [3], and runtime server reconfiguration in CBS using benefit functions [25].

Next, we identify examples of response and utilization accounting in related work dealing with scheduler overhead. In [17], given a set of dynamically scheduled periodic interrupts and tasks, interrupt handler overhead is accounted for in the processor demand of the tasks (related to utilization accounting). In [6], given a set of periodic tasks, so-called explicit overhead through system calls invoked by task code is accounted for in the worst-case execution times of tasks (response accounting) and so-called implicit overhead through scheduler invocations and interrupts is accounted for in CPU utilization (utilization accounting). In [5], given a set of periodic tasks, interrupt and fixed-priority scheduling overhead is accounted for in the response times of the tasks (response accounting). In [9], given a CBS system, scheduler overhead due to suspending processes is

accounted for in response-time bounds of so-called jobs (response accounting). The response-time bounds are tighter than ours exploiting the fact that server parameters cannot be changed and scheduler invocations due to releasing processes are not considered.

In a wider context, examples of previous work dealing with general system overhead are an analysis of event- and time-driven implementations of fixed-priority scheduling [18], algorithms to compute the number of preemptions in sets of periodic, DM- and EDF-scheduled tasks [15], and a study of RM and EDF scheduling that includes a comparison of context-switching overhead [7]. There is also work on reducing context-switching overhead through a modified fixed-priority scheduler [16] and on reducing the number of preemptions in fixed-priority scheduling without modifying the scheduler [14]. The effects of cache-related preemption delays on the execution time of processes are analyzed in [19] and [23].

### 3 VBS scheduling

We have introduced VBS scheduling in previous work [10, 12]. Here, we briefly recall the necessary definitions and results. A variable bandwidth server (VBS) is an extension of a constant bandwidth server (CBS) where throughput and latency of process execution can vary in time under certain restrictions. Given a virtual periodic resource [24], defined as a pair of a period and a limit (with bandwidth equal to the ratio of limit over period), a CBS executes a single process no more than the amount of time given by the resource limit in every time interval given by the resource period. A VBS may vary the virtual periodic resource (change the resource periods and limits), as long as the bandwidth does not exceed a predefined bandwidth cap. A process running on a VBS can initiate a resource switch at any time. The process code from one switch to the next is called a process action. The execution of a process is thus potentially an unbounded sequence of actions. In practice each action can be seen as a piece of sequential code that has a virtual resource associated with it. We implemented a VBS-based scheduling algorithm [11], with four alternative queue management plugins based on lists, arrays, matrices, and trees. The plugins allow trading-off time and space complexity.

#### 3.1 Process Model

A process  $P(u)$  corresponding to a VBS with utilization  $u$  is a finite or infinite sequence of actions,

$$P(u) = \alpha_0 \alpha_1 \alpha_2 \dots$$

for  $\alpha_i \in \text{Act}$ , where  $\text{Act} = \mathbb{N} \times \mathcal{R}$ , with  $\mathcal{R}$  being the finite set of virtual periodic resources [24], explained below. An action  $\alpha \in \text{Act}$  is a pair  $\alpha = (l, R)$  where  $l$  standing for load is a natural number which denotes the exact amount of time the process will perform the action on the virtual periodic resource  $R$ . The load of an action can be understood as the worst-case execution time of the piece

of code that constitutes the action. The virtual periodic resource  $R$  is a pair  $R = (\lambda, \pi)$  of natural numbers with  $\lambda \leq \pi$ , where  $\lambda$  denotes the limit and  $\pi$  the period of the resource. The limit  $\lambda$  specifies the maximum amount of time the process  $P$  can execute on the virtual periodic resource within the period  $\pi$ , while performing the action  $\alpha$ . The utilization of  $R$  is  $u_R = \frac{\lambda}{\pi} \leq u$ .

### 3.2 Schedulability analysis without overhead

Let  $\mathcal{P}$  be a finite set of processes. A schedule for  $\mathcal{P}$  is a partial function

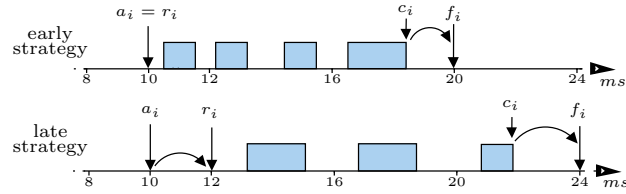
$$\sigma : \mathbb{N} \hookrightarrow \mathcal{P}$$

from the time domain to the set of processes. The function  $\sigma$  assigns to each moment in time a process that is running in the time interval  $[t, t + 1)$ . If no process runs in the interval  $[t, t + 1)$  then  $\sigma(t)$  is undefined. Any scheduler  $\sigma$  of VBS processes determines a unique function  $\sigma_{\mathcal{R}} : \mathbb{N} \hookrightarrow \mathcal{P} \times \mathcal{R}$  that specifies which virtual periodic resource is used by the running process.

We are only interested in *well-behaved* schedules with the property that for any process  $P \in \mathcal{P}$ , any resource  $R \in \mathcal{R}$  with  $R = (\lambda, \pi)$ , and any natural number  $k \in \mathbb{N}$

$$|\{t \in [k\pi, (k+1)\pi) \mid \sigma_{\mathcal{R}}(t) = (P, R)\}| \leq \lambda.$$

Hence, in such a well-behaved schedule, the process  $P$  uses the resource  $R$  at most  $\lambda$  units of time per period of time  $\pi$ . Our scheduling algorithm produces well-behaved schedules.



**Fig. 1.** Scheduling an action  $\alpha_i$  with both release strategies, where  $\lambda_i = 2$  and  $\pi_i = 4$

For each process action  $\alpha_i$  we denote the following absolute moments in time, which are also depicted in Figure 1:

- Arrival time  $a_i$  of the action  $\alpha_i$  is the time instant at which the previous action of the same process has finished. The first action of a process has zero arrival time.
- Completion time  $c_i$  of the action  $\alpha_i$  is the time at which the action completes its execution. It is calculated as

$$c_i = \min \{c \in \mathbb{N} \mid l_i = |\{t \in [a_i, c) \mid \sigma(t) = P\}|\}.$$

- Termination or finishing time  $f_i$  of the action  $\alpha_i$  is the time at which the action terminates or finishes its execution,  $f_i \geq c_i$ . We adopt the following termination strategy: The termination time is at the end of the period within which the action has completed.
- Release time  $r_i$  is the earliest time when  $\alpha_i$  can be scheduled,  $r_i \geq a_i$ . As shown in Figure 1, we consider two release strategies. In the early release strategy  $r_i = a_i$ , i.e., the process is released immediately upon arrival with a fraction of its limit computed for the remaining interval until the period ends. The late release strategy delays the release of an action until the beginning of the next period. The early release strategy may improve average response times.

The scheduled response time  $s_i$  of the action  $\alpha_i$  under the scheduler  $\sigma$  is the difference between the finishing time and the arrival time, i.e.,  $s_i = f_i - a_i$ .

We digress at this point for a brief elaboration on the type of tasks that are generated by VBS processes, which are used to prove the VBS schedulability result.

Let  $\tau = (r, e, d)$  be an aperiodic task with release time  $r$ , execution duration  $e$ , and deadline  $d$ . We say that  $\tau$  has type  $(\lambda, \pi)$  where  $\lambda$  and  $\pi$  are natural numbers,  $\lambda \leq \pi$ , if the following conditions hold:

- $d = (n + 1)\pi$  for a natural number  $n$  such that  $r \in [n\pi, (n + 1)\pi)$ , and
- $e \leq (d - r)\frac{\lambda}{\pi}$ .

The task type  $(\lambda, \pi)$  represents a virtual periodic task which we use in order to impose a bound on the aperiodic task  $\tau$ . If the release of  $\tau$  is at time  $n\pi$ , the execution duration  $e$  is limited by  $\lambda$ . Otherwise, if the release is not at an instance of the period  $\pi$ , the execution duration is adjusted so that the task  $\tau$  has utilization factor at most  $\frac{\lambda}{\pi}$  over the interval  $[r, d]$ .

Let  $S$  be a finite set of task types. Let  $I$  be a finite index set, and consider a set of tasks

$$\{\tau_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j}) \mid i \in I, j \geq 0\}$$

with the properties:

- Each  $\tau_{i,j}$  has a type in  $S$ . We will write  $(\lambda_{i,j}, \pi_{i,j})$  for the type of  $\tau_{i,j}$ .
- The tasks with the same first index are released in a sequence, i.e.,  $r_{i,j+1} \geq d_{i,j}$  and  $r_{i,0} = 0$ .

We refer to such a set as a typed set of tasks.

**Lemma 1 ([12]).** *Let  $\{\tau_{i,j} \mid i \in I, j \geq 0\}$  be a set of tasks as defined above. If*

$$\sum_{i \in I} \max_{j \geq 0} \frac{\lambda_{i,j}}{\pi_{i,j}} \leq 1,$$

*then this set of tasks is schedulable using the EDF strategy at any point of time, so that each task meets its deadline. ■*

The proof of Lemma 1 follows the standard periodic EDF proof of sufficiency with the addition that the periods of the tasks may change in time. The full proof can be found in our previous work on VBS [12]. The same result in a different formulation has also been shown in [4].

Using the definition of VBS processes and actions we can give both response-time bounds for an action and a constant-time schedulability test for an ideal system, i.e., a system without overhead.

Let  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$  be a finite set of  $n$  processes with corresponding actions  $\alpha_{i,j} = (l_{i,j}, R_{i,j})$  for  $j \geq 0$ . Each  $P_i(u_i) = \alpha_{i,0}\alpha_{i,1}\dots$  corresponds to a VBS with utilization  $u_i$ . Let  $R_{i,j} = (\lambda_{i,j}, \pi_{i,j})$  be the virtual periodic resource associated with the action  $\alpha_{i,j}$  with  $l_{i,j}$ ,  $\lambda_{i,j}$ , and  $\pi_{i,j}$  being the load, the limit, and the period for the action  $\alpha_{i,j}$ . The upper bound for the response time of action  $\alpha_{i,j}$  is

$$b_{i,j}^u = \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \pi_{i,j} + \pi_{i,j} - 1.$$

The lower response-time bound varies depending on the strategy used, namely

$$b_{i,j}^l = \begin{cases} \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \pi_{i,j}, & \text{for late release} \\ \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \pi_{i,j}, & \text{for early release.} \end{cases}$$

Note that the lower bound in the early release strategy is achieved only if  $\lambda_{i,j}$  divides  $l_{i,j}$ , in which case  $\left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor = \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil$ . From these bounds we can derive that the response-time jitter, i.e., the difference between the upper and lower bound on the response time, is at most  $\pi_{i,j} - 1$  for the late release strategy and at most  $2\pi_{i,j} - 1$  for the early release strategy. It is possible to give more precise bounds for the early strategy (using (2) and (3) below) and show that also in that case the jitter is at most  $\pi_{i,j} - 1$ .

We say that the set of VBS processes is schedulable with respect to the response-time bounds  $b_{i,j}^u$  and  $b_{i,j}^l$  if there exists a well-behaved schedule  $\sigma$  such that for every action  $\alpha_{i,j}$  the scheduled response time  $s_{i,j}$  satisfies  $b_{i,j}^l \leq s_{i,j} \leq b_{i,j}^u$ .

It is important to note that each action  $\alpha_{i,j}$  of a process produces a sequence of tasks with type  $(\lambda_{i,j}, \pi_{i,j})$  that are released at period instances of the virtual periodic resource used by the action. Scheduling VBS processes therefore amounts to scheduling typed sets of tasks. We call the release of such a task an activation of the action.

**Proposition 1 ([10, 12]).** *Given a set of VBS processes  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$ , if*

$$\sum_{i \in I} u_i \leq 1,$$

*then the set of processes  $\mathcal{P}$  is schedulable with respect to the response-time bounds  $b_{i,j}^u$  and  $b_{i,j}^l$ .*

*Proof.* As mentioned above, each process  $P_i$  provides a typed set of tasks since each action  $\alpha_{i,j}$  of the process is split into several tasks that all have the type  $(\lambda_{i,j}, \pi_{i,j})$ . The tasks are released in a sequence, i.e. the release time of the next task is always equal or greater than the deadline of the current task (also due to the termination strategy). We can thus apply Lemma 1 and conclude that the set of tasks is schedulable. To check the bounds, we distinguish a case for the early release strategy and one for the late release strategy in terms of the first task of an action. The details can be found in [12]. We present the proof for the lower bounds which is not discussed in [12]. For each action  $\alpha_{i,j}$ , according to the termination strategy and the late release strategy, we have

$$f_{i,j} = r_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \pi_{i,j} \quad (1)$$

where the release times are given by  $r_{i,j} = n_j \pi_{i,j}$  for some natural number  $n_j$  such that the arrival times are  $a_{i,j} = f_{i,j-1} \in ((n_j - 1)\pi_{i,j}, n_j \pi_{i,j}]$ . Therefore, for the late strategy we have

$$\begin{aligned} s_{i,j} &= f_{i,j} - a_{i,j} \stackrel{(1)}{=} \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \pi_{i,j} + r_{i,j} - a_{i,j} \\ &\geq \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \pi_{i,j} \\ &= b_{i,j}^l, \text{ for late release.} \end{aligned}$$

For the early release strategy we distinguish two cases depending on whether the following inequality holds

$$\left\lfloor m \frac{\lambda_{i,j}}{\pi_{i,j}} \right\rfloor \geq l_{i,j} - \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \lambda_{i,j} \quad (2)$$

where  $m = n_j \pi_{i,j} - r_{i,j}$  and  $r_{i,j} = a_{i,j} = f_{i,j-1} \in ((n_j - 1)\pi_{i,j}, n_j \pi_{i,j}]$ . The finishing time for the action  $\alpha_{i,j}$  is

$$f_{i,j} = \begin{cases} a_{i,j} + \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \pi_{i,j} + m, & \text{if (2) holds} \\ a_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \pi_{i,j} + m, & \text{otherwise} \end{cases} \quad (3)$$

In both cases  $f_{i,j} \geq \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \pi_{i,j} + a_{i,j}$ , so

$$\begin{aligned} s_{i,j} &= f_{i,j} - a_{i,j} \\ &\geq \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \pi_{i,j} \\ &= b_{i,j}^l, \text{ for early release.} \end{aligned}$$

■

In the following sections we analyze VBS schedulability in the presence of scheduler overhead. In particular, we define an upper bound on the number of scheduler invocations that occur within a period of an action, and perform an analysis of the changes in the response-time bounds and utilization due to the scheduler overhead.

## 4 VBS scheduling with overhead

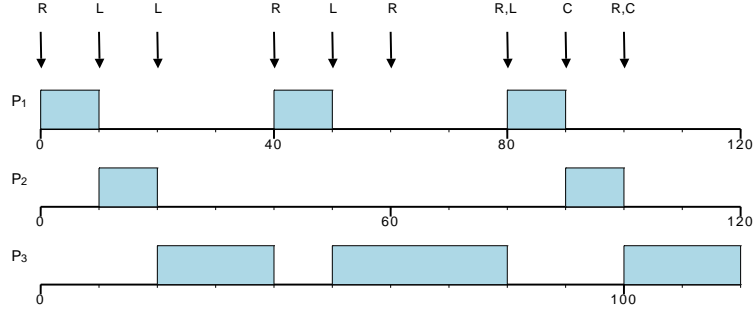
Our first aim is to bound the number of scheduler invocations over a time interval. In particular, we need the worst-case number of preemptions, and hence scheduler invocations, that an action of a VBS process experiences during a period. Hence, by overhead we mean the overhead due to scheduler invocations (and not interrupts or other types of system overhead).

Typically the duration of a scheduler invocation is several orders of magnitude lower than a unit execution of an action. Therefore, we assume that all periods belong to the set of discrete time instants  $M = \{c \cdot n \mid n \geq 0\} \subset \mathbb{N}$ , for a constant value  $c \in \mathbb{N}, c > 1$ . Hence, for any action  $\alpha_{i,j}$  with its associated virtual periodic resource  $R_{i,j} = (\lambda_{i,j}, \pi_{i,j})$  we have that  $\pi_{i,j} = c \cdot \pi'_{i,j}$  with  $\pi'_{i,j} \in \mathbb{N}$ . We call  $c$  the scale of the system. Intuitively we can say that there are two different timelines, the “fine-grained timeline” given by the set of natural numbers and the “coarse-grained timeline” given by the set  $M$ . Resource periods are defined on the “coarse-grained timeline”, while the execution time of the scheduler is defined on the “fine-grained timeline”.

In VBS scheduling, a process  $P_i$  is preempted at a time instant  $t$  if and only if one of the following situations occurs:

1. Completion.  $P_i$  has completed the entire work related to its current action  $\alpha_{i,j} = (l_{i,j}, R_{i,j})$ .
2. Limit.  $P_i$  uses up all resource limit  $\lambda_{i,j}$  of the current resource  $R_{i,j}$ .
3. Release. A task of an action is released at time  $t$ , i.e., an action of another process is activated. Note that all preemptions due to release occur at time instants on the “coarse-grained timeline”, the set  $M$ .

*Example 1.* Consider the first action of three processes  $P_1, P_2,$  and  $P_3,$  with corresponding virtual periodic resources  $R_{1,0} = (\lambda_{1,0}, \pi_{1,0}) = (10, 40), R_{2,0} = (\lambda_{2,0}, \pi_{2,0}) = (10, 60),$  and  $R_{3,0} = (\lambda_{3,0}, \pi_{3,0}) = (50, 100),$  and loads 30, 20, and 100, respectively. Figure 2 shows an example schedule of the three actions up to time 120 and the time instants at which preemptions occur. Preemptions that occur because of the release of an action instance (action activation) are labeled with R, preemptions that occur because an action has finished its limit are labeled with L, and preemptions that are due to an action finishing its entire load have the label C. At time 0, 40, 60, 80, and 100 preemptions occur due to an activation of one of the three actions, while at time 10, 20, 50, and 80 preemptions occur because an action has used all its limit for the current period. At times 90 and 100 preemptions occur due to completion of an action. At certain time



**Fig. 2.** Example schedule for processes  $P_1$ ,  $P_2$ , and  $P_3$

instants such as 80 and 100 there is more than one reason for preemption. In this example the scale  $c$  of the system is 10.

Let us first consider preemptions due to release, i.e., activation of an action. Each activation happens when a new period of the used virtual periodic resource starts. Hence, an action  $\alpha_{i,j}$  with a virtual periodic resource  $(\lambda_{i,j}, \pi_{i,j})$  will only be activated at time instants  $k \cdot \pi_{i,j}$  with  $k \in \mathbb{N}$ . Therefore, in the worst-case scenario, the preemptions caused by action activations of all processes in the system occur at all time instants in the set  $\{k \cdot \gcd(\{\pi_{i,j} \mid i \in I, j \geq 0\}) \mid k \in \mathbb{N}\}$ . Note that, because the periods of all actions in the system are on the “coarse-grained timeline”,  $\gcd(\{\pi_{i,j} \mid i \in I, j \geq 0\}) \geq c$ .

We compute the scheduler overhead for each action using an upper bound on the number of preemptions during one period of the respective action.

**Lemma 2.** *Let  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$  be a set of VBS processes with actions  $\alpha_{i,j}$  and corresponding virtual periodic resources  $(\lambda_{i,j}, \pi_{i,j})$ . There are at most*

$$N_{i,j} = N_{i,j}^R + N_{i,j}^L$$

*scheduler invocations every period  $\pi_{i,j}$  for the action  $\alpha_{i,j}$ , where*

$$N_{i,j}^R = \left\lceil \frac{\pi_{i,j}}{\gcd(\{\pi_{m,n} \mid m \in I, n \geq 0, m \neq i\})} \right\rceil \quad (4)$$

*and  $N_{i,j}^L = 1$ .*

*Proof.* As discussed above, there are scheduler invocations at most at every  $k \cdot \gcd(\{\pi_{m,n} \mid m \in I, n \geq 0\})$ ,  $k \in \mathbb{N}$  due to release. For an action  $\alpha_{i,j}$  with period  $\pi_{i,j}$ , belonging to a process  $P_i(u)$ , the number of preemptions due to release of other processes does not depend on its own period nor on the periods of the other actions of the same process. Hence, there can be at most  $N_{i,j}^R$  preemptions due to release over a period of the considered action. By design of the scheduling algorithm, there is exactly one more scheduler invocation due to the action using all its limit or completing its entire load in each period, i.e.,  $N_{i,j}^L = 1$ .  $\blacksquare$

This is a pessimistic approximation which could be improved by running a runtime analysis that considers only the periods of the active actions at a specific time. Nevertheless, we take the pessimistic approach in order to be able to analyze the system off-line.

Another bound on the number of preemptions due to release for sets of periodic tasks has been given in [6]. For an action  $\alpha_{i,j}$ , it calculates the number of period instances from other actions that occur during the period  $\pi_{i,j}$ . The worst-case number of scheduler invocations for a period  $\pi_{i,j}$  of an action due to release is thus

$$\sum_{\substack{k \in I \\ k \neq i}} \sum_{l > 0} \left\lceil \frac{\pi_{i,j}}{\pi_{k,l}} \right\rceil. \quad (5)$$

Depending on the periods in the system one of the two given bounds approximates the worst-case number better than the other. Consider for example a case where the periods in the system are 3, 5, and 7. In this case (5) approximates the number of preemptions better than (4). In another example, if the periods in the system are 2, 4, and 6, then (4) provides a more accurate bound than (5). We choose to consider (4) for a bound on the number of preemptions due to release, for reasons that will be explained in Section 6.

Case	Overhead distribution	Load
RA	$\delta_{i,j}^b = \delta_{i,j}, \delta_{i,j}^u = 0$	$l_{i,j}^* = l_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j} - \delta_{i,j}} \right\rceil \delta_{i,j}$
UA	$\delta_{i,j}^b = 0, \delta_{i,j}^u = \delta_{i,j}$	$l_{i,j}^* = l_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \delta_{i,j}$
RUA	$\delta_{i,j}^b, \delta_{i,j}^u > 0$	$l'_{i,j} = l_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j} - \delta_{i,j}^b} \right\rceil \delta_{i,j}^b, l_{i,j}^* = l'_{i,j} + \left\lceil \frac{l'_{i,j}}{\lambda_{i,j}} \right\rceil \delta_{i,j}^u$

Case	Utilization	Schedulability test
RA	$\frac{\lambda_{i,j}}{\pi_{i,j}}$	$\sum_{i \in I} \max_{j \geq 0} \frac{\lambda_{i,j}}{\pi_{i,j}} \leq 1$
UA	$\frac{\lambda_{i,j} + \delta_{i,j}}{\pi_{i,j}}$	$\sum_{i \in I} \max_{j \geq 0} \frac{\lambda_{i,j} + \delta_{i,j}}{\pi_{i,j}} \leq 1$
RUA	$\frac{\lambda_{i,j} + \delta_{i,j}^u}{\pi_{i,j}}$	$\sum_{i \in I} \max_{j \geq 0} \frac{\lambda_{i,j} + \delta_{i,j}^u}{\pi_{i,j}} \leq 1$

Table 1. Scheduler overhead accounting

## 5 Schedulability analysis with overhead

In a real system the scheduler overhead manifests itself as additional load which must be accounted for within the execution of a process in order not to invalidate the schedule of the system. Let  $\xi$  denote the duration of a single scheduler

invocation. The total scheduler overhead for one period of action  $\alpha_{i,j}$  is therefore

$$\delta_{i,j} = N_{i,j} \cdot \xi.$$

Hence, the total overhead is made up of  $N_{i,j}$  pieces of  $\xi$  workload. An important aspect in the analysis is that a scheduler invocation with overhead  $\xi$  is nonpreemptable.

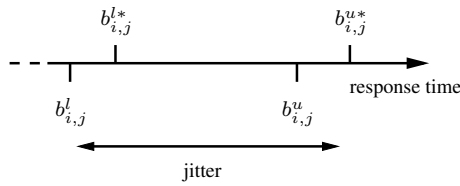
Accounting for the overhead can be done in two ways. One way is to allow an action to execute for less time than its actual limit within one period and use the remaining time to account for the scheduler overhead. The other way is to increase the limit such that the action will be able to execute its original limit and the time spent on scheduler invocations within one period. Intuitively, the first method increases the response-time bounds, and the second increases the utilization of an action. We recognize a fundamental trade-off between an increase in response time versus utilization by distributing the amount of scheduler overhead. Namely, we write that the overhead is

$$\delta_{i,j} = \delta_{i,j}^b + \delta_{i,j}^u,$$

where  $\delta_{i,j}^b$  is the overhead that extends the response-time bounds of the respective action and  $\delta_{i,j}^u$  increases the utilization. Note that no scheduler invocation is divisible, i.e., both  $\delta_{i,j}^b$  and  $\delta_{i,j}^u$  are multiples of  $\xi$ .

There are three cases:

- Response accounting (RA),  $\delta_{i,j} = \delta_{i,j}^b$ , when the entire overhead is executing within the limit of the action, keeping both the limit and period (and thus the utilization) of the actions constant but increasing the response-time bounds.
- Utilization accounting (UA),  $\delta_{i,j} = \delta_{i,j}^u$ , when the entire overhead increases the limit of the action, and thus the utilization, but the response-time bounds remain the same.
- Combined accounting (RUA), with  $\delta_{i,j} = \delta_{i,j}^b + \delta_{i,j}^u$ ,  $\delta_{i,j}^b > 0$ , and  $\delta_{i,j}^u > 0$ , which offers the possibility to trade-off utilization for response time, for each action, in the presence of scheduler overhead.



**Fig. 3.** Bounds and jitter with and without overhead

For an action  $\alpha_{i,j}$ , in the presence of overhead, we denote the new load by  $l_{i,j}^*$ , the new limit by  $\lambda_{i,j}^*$ , and the new utilization by  $u_{i,j}^*$ . Using these new

parameters for an action we determine the new upper and lower response-time bounds which we denote with  $b_{i,j}^{u*}$  and  $b_{i,j}^{l*}$ , respectively. The upper response-time bound  $b_{i,j}^{u*}$  for action  $\alpha_{i,j}$  is

$$b_{i,j}^{u*} = \left\lceil \frac{l_{i,j}^*}{\lambda_{i,j}^*} \right\rceil \pi_{i,j} + \pi_{i,j} - 1. \quad (6)$$

The lower response-time bound  $b_{i,j}^{l*}$  for  $\alpha_{i,j}$  using the late release strategy is

$$b_{i,j}^{l*} = \left\lceil \frac{l_{i,j}^*}{\lambda_{i,j}^*} \right\rceil \pi_{i,j}, \quad (7)$$

whereas using the early release strategy is

$$b_{i,j}^{l*} = \left\lfloor \frac{l_{i,j}^*}{\lambda_{i,j}^*} \right\rfloor \pi_{i,j}. \quad (8)$$

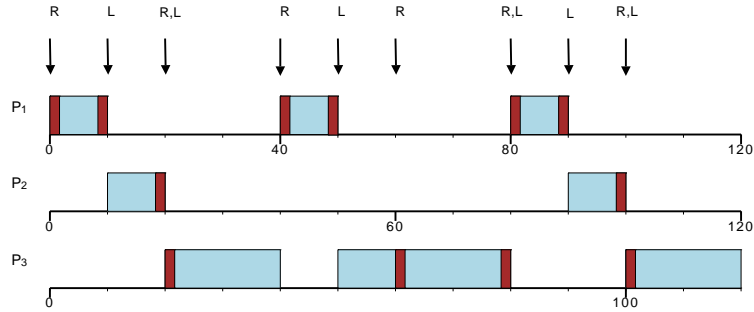
In the previous section we have given an upper bound on the number of preemptions that an action will experience during one period. This number is used to compute the scheduler overhead and hence the new response-time bounds for the respective action. As shown in Figure 3 and discussed in the remainder of this section, both the upper and the lower bounds increase if we assume that the worst-case number of preemptions actually occurs. However, our analysis does not provide information on the actual number of scheduler invocations during one period. Hence, when determining the upper bound on response-time jitter, the assumption that an action always experiences the worst-case number of preemptions is invalid. Therefore, the jitter is at most the difference between the new upper response-time bound  $b_{i,j}^{u*}$  and the ideal lower response-time bound  $b_{i,j}^l$ , as shown in Figure 3.

Table 1 summarizes the three cases which we discuss in detail in the remainder of this section. We elaborate on the change in response-time bounds, utilization, and jitter for each case.

## 5.1 Response Accounting

In the response accounting case each action executes for less time than its actual limit so that enough time remains for the scheduler execution. As a result, the action will not execute more than its limit even when scheduler overhead is considered. In this way the utilization of the action remains the same but its response-time bounds increase. We have that  $\delta_{i,j} = \delta_{i,j}^b$ .

*Example 2.* Consider the same processes as in Example 1. The schedule in Figure 4 shows how the scheduler overhead can be accounted for. At each scheduler invocation due to release (labeled with R), the process that is scheduled to run accounts for the overhead. Note that in our analysis, unlike in this example



**Fig. 4.** Example of response accounting of scheduler overhead

schedule, we account for an over-approximation of the number of scheduler invocations due to release. In the case of preemptions due to limit or completion (labeled with L or C) the preempted process accounts for the overhead. This implies that in case of preemptions due to limit, the action has to be preempted before the limit is exhausted so that there is time for the scheduler to execute within the limit.

The response-time bounds for each action differ from the ones given in Section 3.2 due to the fact that each action can execute less of its own load in the presence of scheduler overhead. We compute the new load of the action  $\alpha_{i,j}$  as

$$l_{i,j}^* = l_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j} - \delta_{i,j}} \right\rceil \delta_{i,j}.$$

An obvious condition for the system to be feasible is  $\delta_{i,j} < \lambda_{i,j}$ . Intuitively, if  $\delta_{i,j} = \lambda_{i,j}$ , the action  $\alpha_{i,j}$  would make no progress within one period as it will just be executing the scheduler, and hence it will never complete its load. If  $\delta_{i,j} > \lambda_{i,j}$ , given that the execution of the scheduler is nonpreemptable,  $\alpha_{i,j}$  will exceed its limit  $\lambda_{i,j}$ , which may result in process actions missing their deadlines or processes not being schedulable anymore. The new limit and utilization of the action are the same as without overhead, i.e.  $\lambda_{i,j}^* = \lambda_{i,j}$  and  $u_{i,j}^* = u_{i,j} = \frac{\lambda_{i,j}}{\pi_{i,j}}$ . Since  $l_{i,j}^* > l_{i,j}$  and  $\lambda_{i,j}^* = \lambda_{i,j}$ , we get that both the upper and the lower response-time bound increases in case of response accounting.

**Proposition 2.** *Let  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$  be a set of VBS processes each with bandwidth cap  $u_i$ . If*

$$\sum_{i \in I} u_i \leq 1 \quad \text{and} \quad \delta_{i,j} < \lambda_{i,j},$$

*with  $\delta_{i,j}, \lambda_{i,j}$  as defined above, then the set of processes are schedulable with respect to the new response-time bounds  $b_{i,j}^{u,*}$  and  $b_{i,j}^{l,*}$ , in the presence of worst-case scheduler overhead.*

*Proof.* The proof follows from Proposition 1, and the discussion above, when substituting the load  $l_{i,j}$  with the new load  $l_{i,j}^*$ . ■

The jitter for any action  $\alpha_{i,j}$  in the response accounting case is at most  $b_{i,j}^{u^*} - b_{i,j}^l$ .

For further reference, we write the new load in the response accounting case as a function

$$RA(l, \lambda, \delta) = l + \left\lceil \frac{l}{\lambda - \delta} \right\rceil \delta.$$

## 5.2 Utilization Accounting

In the utilization accounting case an action is allowed to execute for more time than its original limit within a period in order to account for scheduler overhead. Thus, we have that  $\delta_{i,j} = \delta_{i,j}^u$ . The new load of action  $\alpha_{i,j}$  becomes

$$l_{i,j}^* = l_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \delta_{i,j}.$$

The new limit is  $\lambda_{i,j}^* = \lambda_{i,j} + \delta_{i,j}$ , and the new utilization is

$$u_{i,j}^* = \frac{\lambda_{i,j} + \delta_{i,j}}{\pi_{i,j}}.$$

**Proposition 3.** *Given a set of processes  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$ , let*

$$u_i^* = \max_{j \geq 0} \frac{\lambda_{i,j} + \delta_{i,j}}{\pi_{i,j}}.$$

*If*

$$\sum_{i \in I} u_i^* \leq 1,$$

*then the set of processes  $\mathcal{P}$  is schedulable with respect to the original response-time bounds  $b_{i,j}^u$  and  $b_{i,j}^l$  defined in Section 3.2, in the presence of worst-case scheduler overhead.*

*Proof.* We consider a modified set of processes  $\mathcal{P}^* = \{P_i^*(u_i^*) \mid 1 \leq i \leq n\}$ . By Proposition 1, this set of processes is schedulable with respect to the response-time bounds  $b_{i,j}^{u^*}$  and  $b_{i,j}^{l^*}$  calculated using the new load  $l_{i,j}^*$  and the new limit  $\lambda_{i,j}^*$ . We will prove that the upper response-time bounds with the new load and limit for action  $\alpha_{i,j}$  are the same as the original upper bounds without overhead ( $b_{i,j}^{u^*} = b_{i,j}^u$ ), and the new lower bound is not lower than the old lower bound ( $b_{i,j}^{l^*} \geq b_{i,j}^l$ ). We start by showing that  $\left\lceil \frac{l_{i,j}^*}{\lambda_{i,j}^*} \right\rceil = \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil$ .

Let  $d \in \mathbb{R}$  be the difference

$$d = \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil - \frac{l_{i,j}^*}{\lambda_{i,j}^*}.$$

It suffices to establish that  $d \in [0, 1)$  in order to prove our claim. We have,

$$d = \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor - \frac{l_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \delta_{i,j}}{\lambda_{i,j} + \delta_{i,j}} = \frac{\left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \lambda_{i,j} - l_{i,j}}{\lambda_{i,j} + \delta_{i,j}}.$$

Both  $\left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \lambda_{i,j} - l_{i,j} \geq 0$  and  $\lambda_{i,j} + \delta_{i,j} > 0$ , so  $d \geq 0$ .

If  $\left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor = \frac{l_{i,j}}{\lambda_{i,j}}$ , then  $d = 0$  and we are done. If  $\left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \neq \frac{l_{i,j}}{\lambda_{i,j}}$ , we can write

$$l_{i,j} = \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \lambda_{i,j} + \Delta = \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \lambda_{i,j} - (\lambda_{i,j} - \Delta),$$

for  $0 < \Delta < \lambda_{i,j}$ . Therefore,

$$d = \frac{\lambda_{i,j} - \Delta}{\lambda_{i,j} + \delta_{i,j}} < \frac{\lambda_{i,j}}{\lambda_{i,j} + \delta_{i,j}} < 1.$$

Hence we have established  $\left\lfloor \frac{l_{i,j}^*}{\lambda_{i,j}^*} \right\rfloor = \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor$ . We then show that  $\left\lfloor \frac{l_{i,j}^*}{\lambda_{i,j}^*} \right\rfloor \geq \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor$ . There are three cases to consider:

1. If  $\lambda_{i,j}$  does not divide  $l_{i,j}$ , and  $\lambda_{i,j}^*$  does not divide  $l_{i,j}^*$ , then  $\left\lfloor \frac{l_{i,j}^*}{\lambda_{i,j}^*} \right\rfloor = \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor$ .
2. If  $\lambda_{i,j}$  does not divide  $l_{i,j}$ , but  $\lambda_{i,j}^*$  divides  $l_{i,j}^*$ , then  $\left\lfloor \frac{l_{i,j}^*}{\lambda_{i,j}^*} \right\rfloor = \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil > \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor$ .
3. The case when  $\lambda_{i,j}$  divides  $l_{i,j}$ , but  $\lambda_{i,j}^*$  does not divide  $l_{i,j}^*$  is not possible, since as we saw before, if  $\lambda_{i,j}$  divides  $l_{i,j}$ , then  $\lambda_{i,j}^*$  divides  $l_{i,j}^*$ .

Hence, the set of processes is schedulable with respect to the old upper bound and the new lower bound (which is greater than or equal to the old lower bound), which makes it schedulable with respect to the old bounds.  $\blacksquare$

In the utilization accounting case, the jitter for any action is the same as in the analysis without overhead, because the response-time bounds are the same.

We write the new load again as a function

$$UA(l, \lambda, \delta) = l + \left\lceil \frac{l}{\lambda} \right\rceil \delta.$$

### 5.3 Combined Accounting

In the combined accounting case, both the response-time bounds and the utilization of an action increase. We have that  $\delta_{i,j} = \delta_{i,j}^b + \delta_{i,j}^u$ ,  $\delta_{i,j}^b > 0$ , and  $\delta_{i,j}^u > 0$ . Given an action  $\alpha_{i,j}$  with its associated virtual periodic resource  $R_{i,j} = (\lambda_{i,j}, \pi_{i,j})$ , and load  $l_{i,j}$ , the new load  $l_{i,j}^*$  is computed in two steps. First we account for the overhead that increases the response time

$$l'_{i,j} = l_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j} - \delta_{i,j}^b} \right\rceil \delta_{i,j}^b$$

and then we add the overhead that increases the utilization

$$l_{i,j}^* = l'_{i,j} + \left\lceil \frac{l'_{i,j}}{\lambda_{i,j}} \right\rceil \delta_{i,j}^u.$$

The load function for the combined case is therefore

$$RUA(l, \lambda, \delta^b, \delta^u) = UA(RA(l, \lambda, \delta^b), \lambda, \delta^u).$$

The new limit for action  $\alpha_{i,j}$  is  $\lambda_{i,j}^* = \lambda_{i,j} + \delta_{i,j}^u$ , and the utilization becomes

$$u_{i,j}^* = \frac{\lambda_{i,j} + \delta_{i,j}^u}{\pi_{i,j}}.$$

The upper response-time bound  $b_{i,j}^{u*}$  for action  $\alpha_{i,j}$  is now

$$b_{i,j}^{u*} = \left\lceil \frac{RUA(l_{i,j}, \lambda_{i,j}, \delta_{i,j}^b, \delta_{i,j}^u)}{\lambda_{i,j} + \delta_{i,j}^u} \right\rceil \pi_{i,j} + \pi_{i,j} - 1.$$

The lower response-time bound  $b_{i,j}^{l*}$  for the same action using the late release strategy is

$$b_{i,j}^{l*} = \left\lceil \frac{RUA(l_{i,j}, \lambda_{i,j}, \delta_{i,j}^b, \delta_{i,j}^u)}{\lambda_{i,j} + \delta_{i,j}^u} \right\rceil \pi_{i,j},$$

and using the early release strategy is

$$b_{i,j}^{l*} = \left\lfloor \frac{RUA(l_{i,j}, \lambda_{i,j}, \delta_{i,j}^b, \delta_{i,j}^u)}{\lambda_{i,j} + \delta_{i,j}^u} \right\rfloor \pi_{i,j}.$$

**Proposition 4.** *Given a set of processes  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$ , let*

$$u_i^* = \max_{j \geq 0} \frac{\lambda_{i,j} + \delta_{i,j}^u}{\pi_{i,j}}.$$

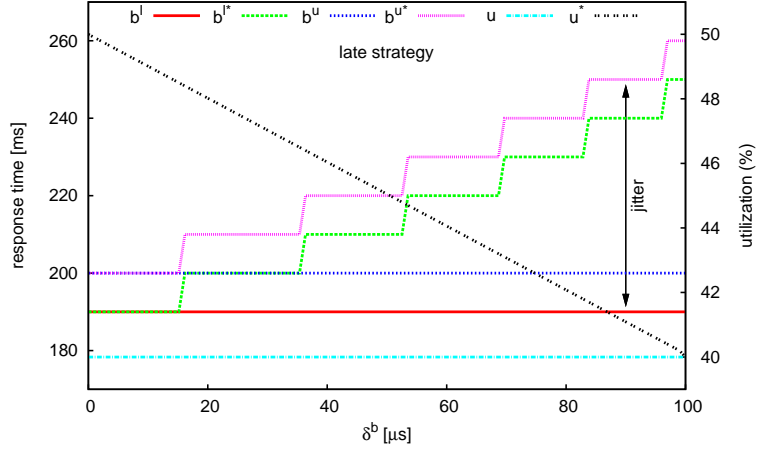
*If*

$$\sum_{i \in I} u_i^* \leq 1,$$

*then the set of processes  $\mathcal{P}$  is schedulable with respect to the response-time bounds  $b_{i,j}^{u*}$  and  $b_{i,j}^{l*}$ , in the presence of worst-case scheduler overhead.*

*Proof.* This schedulability result is derived by combining Proposition 2 and Proposition 3 in the response accounting and utilization accounting case, respectively. ■

Figure 5 shows the effect of the scheduler overhead distribution on the response time and utilization for an example action. We consider the action  $\alpha$  with limit  $\lambda = 400\mu s$ , period  $\pi = 1000\mu s$ , and load  $l = 7300\mu s$ . The total scheduler overhead of  $\delta = 100\mu s$  corresponds to 100 scheduler invocations, each



**Fig. 5.** Response time and utilization with  $\delta^b$  varying in  $[0\mu s, 100\mu s]$  for  $\alpha = (7300\mu s, (400\mu s, 1000\mu s))$  using the late strategy

with overhead  $\xi = 1\mu s$ . If  $\delta^b = 0$  (utilization accounting), the lower and upper response-time bounds  $b^{l*}$  and  $b^{u*}$  remain the same as the respective bounds  $b^l$  and  $b^u$  without scheduler overhead accounting but the utilization  $u^*$  is 10% higher than the utilization  $u$  without scheduler overhead accounting. If  $\delta^b = \delta$  (response accounting), we have that  $u^* = u$  but  $b^{l*}$  and  $b^{u*}$  are several periods larger than  $b^l$  and  $b^u$ , respectively, also resulting in a larger bound  $b^{u*} - b^l$  on the response-time jitter. As  $\delta^b$  increases,  $u^*$  decreases while  $b^{u*}$  increases along with  $b^{l*}$  and the jitter bound. Note that, depending on the involved parameters, we can change the overhead distribution within some intervals such that the utilization decreases but the response-time bounds remain the same, for example, when  $\delta^b$  varies between  $0\mu s$  and  $16\mu s$ .

## 6 Optimization

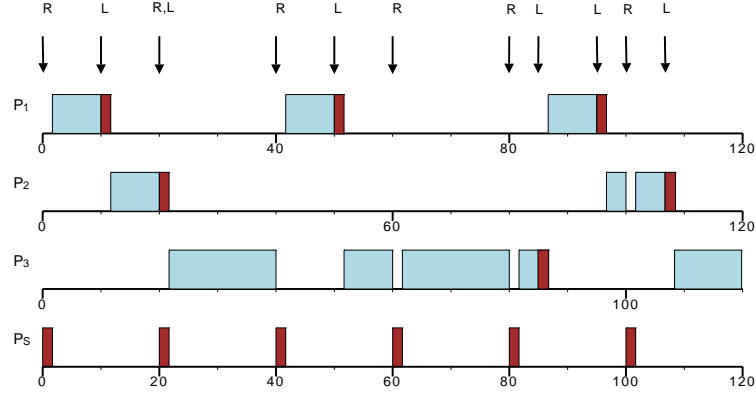
There is a natural division of the VBS scheduler overhead into overhead due to release  $\delta_{i,j}^R$  and overhead due to limit/completion  $\delta_{i,j}^L$ . In Section 4 we bounded the overhead due to release for each period of an action as  $\delta_{i,j}^R = N_{i,j}^R \cdot \xi$ . In this estimate, each action accounts for the release of each of the other actions in the system, or in other words, when an action is activated all other processes account for the overhead, although at most one process is truly preempted. This is clearly an over-approximation. As already discussed, all preemptions due to release occur at time instants given by the set  $\{k \cdot \gcd(\{\pi_{i,j} \mid i \in I, j \geq 0\}) \mid k \in \mathbb{N}\}$ . Hence, instead of letting other processes account for the release overhead, the overhead can be modeled as a separate, virtual VBS process with

Case	Overhead distribution	Load	
RA	$\delta_{i,j}^b = \delta_{i,j}^L + \delta_{i,j}^R, \delta_{i,j}^u = 0$	$l_{i,j}^* = l_{i,j} + \frac{l_{i,j}}{\lambda_{i,j} - \delta_{i,j}}$	$\delta_{i,j}$
UA	$\delta_{i,j}^b = 0, \delta_{i,j}^u = \delta_{i,j}^L + \delta_{i,j}^R$	$l_{i,j}^* = l_{i,j} + \frac{l_{i,j}}{\lambda_{i,j}}$	$\xi$
RUA	$\delta_{i,j}^b = \delta_{i,j}^L, \delta_{i,j}^u = \delta_{i,j}^R$	$l_{i,j}^* = l_{i,j} + \frac{l_{i,j}}{\lambda_{i,j} - \xi}$	$\xi$

Case	Utilization	Schedulability test
RA	$\frac{\lambda_{i,j}}{\pi_{i,j}}$	$\sum_{i \in I} \max_{j \geq 0} \frac{\lambda_{i,j}}{\pi_{i,j}} \leq 1$
UA	$\frac{\lambda_{i,j} + \xi}{\pi_{i,j}}$	$\sum_{i \in I} \max_{j \geq 0} \frac{\lambda_{i,j} + \xi}{\pi_{i,j}} \leq 1 - u_S$
RUA	$\frac{\lambda_{i,j}}{\pi_{i,j}}$	$\sum_{i \in I} \max_{j \geq 0} \frac{\lambda_{i,j}}{\pi_{i,j}} \leq 1 - u_S$

**Table 2.** Cases with optimization

the same action repeated infinitely many times. We call this process the scheduler process. Introducing a virtual process allows accounting for the overhead due to release only once, and not (as before) in every process. As a result, some of the presented scheduling results optimize, by weakening the utilization-based schedulability test. Note that the scheduler process provides an optimization only if the scheduler invocations due to release are accounted for in the utilization of the processes, i.e., that  $\delta_{i,j}^u \geq \delta_{i,j}^R$ .



**Fig. 6.** Example of scheduler overhead as a separate, virtual VBS process  $P_S$  for the combined accounting case

In addition to a set of VBS processes (as before) there is the scheduler process  $P_S$  with all actions equal  $\alpha_{S,j} = (\xi, R_S)$  where  $R_S = (\lambda_S, \pi_S) = (\xi, \gcd(\{\pi_{i,j} \mid$

$i \in I, j \geq 0\}$ ). Thus, the utilization of the scheduler process is

$$u_S = \frac{\xi}{\gcd(\{\pi_{i,j} \mid i \in I, j \geq 0\})}.$$

Note that the scheduler process accounts only for the preemptions due to the release of an action but not for the actions using all their limit or completing their load.

*Example 3.* Consider the same processes as in Example 1. The schedule in Figure 6 shows how the scheduler overhead resulting from releases is integrated into the scheduler process. In cases where preemption can occur due to multiple reasons, such as at time 20, we prioritize the release and let the scheduler process run. As can be seen in Figure 6 and Figure 4, the scheduler process may not optimize the actual schedule, but it does optimize the schedulability test.

Table 2 summarizes the effect of the optimization to all cases. Note that the response accounting case cannot be optimized, since  $\delta_{i,j}^u = 0$  in this case.

### 6.1 Combined accounting

In the combined accounting case an optimization is possible only in the case  $\delta_{i,j}^u = \delta_{i,j}^R$  and  $\delta_{i,j}^b = \delta_{i,j}^L = \xi$ . The overhead due to limit/completion continues to add to the response time of each process and thus the new load for action  $\alpha_{i,j}$  is

$$l_{i,j}^* = l_{i,j} + \left\lceil \frac{l_{i,j}}{\lambda_{i,j} - \xi} \right\rceil \xi$$

and the utilization remains  $\frac{\lambda}{\pi}$ . The bounds  $b_{i,j}^{u*}$  and  $b_{i,j}^{l*}$  are given by (6), (7) and (8), as before. The next result presents an optimization to Proposition 4.

**Proposition 5.** *Given a set of processes  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$ . If*

$$\sum_{i \in I} u_i \leq 1 - u_S,$$

*then the set of processes  $\mathcal{P}$  is schedulable with respect to the response-time bounds  $b_{i,j}^{u*}$  and  $b_{i,j}^{l*}$ , in the presence of worst-case scheduler overhead.*

*Proof.* Let  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$  be a set of VBS processes each with bandwidth cap  $u_i$ . Let  $P_S$  be the scheduler process as defined above. It is important to note that the scheduler process generates tasks that always have the highest priority when released. This is because the action  $\alpha_{S,k}$  has the period  $\pi_S = \gcd(\{\pi_{i,j} \mid i \in I, j \geq 0\}) \leq \pi_{i,j}, \forall i \in I, \forall j \geq 0$ . Hence, at every invocation time of the action its deadline is earlier or at the same time as any other deadline in the system and thus we can be sure it is never preempted by any other task using EDF. By Proposition 4 we get that the extended set of processes (together with the scheduler process) is schedulable with respect to the new bounds if

$$u_S + \sum_{i \in I} \max_{j \geq 0} \frac{\lambda_{i,j}}{\pi_{i,j}} \leq 1.$$

■

## 6.2 Utilization accounting

In the case when the entire overhead increases the utilization of an action, optimization is possible. For the preemptions due to limit or completion, the limit of the action  $\alpha_{i,j}$  becomes  $\lambda_{i,j}^* = \lambda_{i,j} + \xi$  and therefore the utilization is

$$u_{i,j}^* = \frac{\lambda_{i,j} + \xi}{\pi_{i,j}}.$$

Note that it is not possible to account for such preemptions in the scheduler process as that would mean that the scheduler process could execute at every time instant of the “fine-grained timeline”. The next result presents the optimized version of Proposition 3.

**Proposition 6.** *Given a set of processes  $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$ , let*

$$u_i^* = \max_{j \geq 0} \frac{\lambda_{i,j} + \xi}{\pi_{i,j}}.$$

If

$$\sum_{i \in I} u_i^* \leq 1 - u_S,$$

then the set of processes  $\mathcal{P}$  is schedulable with respect to the response-time bounds  $b_{i,j}^u$  and  $b_{i,j}^l$  defined in Section 3.2, in the presence of worst-case scheduler overhead.

*Proof.* The proof is similar to Proposition 3 and Proposition 5 with the difference that at any time the system consists of tasks with the type  $(\lambda_{i,j} + \xi, \pi_{i,j})$  generated by the processes in  $\mathcal{P}^*$  and the tasks generated by the scheduler process. The response-time bounds are proven to remain unchanged by substituting  $\delta_{i,j}$  with  $\xi$  in Proposition 3.  $\blacksquare$

Clearly, the set of processes is not schedulable if  $\xi \geq \gcd(\{\pi_{i,j} \mid i \in I, j \geq 0\})$ . The utilization that can be used by processes drops to 0 if  $\xi = c$ . The system is thus better suited for values of  $c$  that respect the condition  $\xi \ll c \leq \gcd(\{\pi_{i,j} \mid i \in I, j \geq 0\})$ .

## 7 Scheduler overhead as a function

A bare-metal implementation of a VBS-based scheduling algorithm, with three different plugins that allow trading-off time and space complexity, exists and has been presented in [10, 12]. Table 3 shows the time and space complexities of the scheduler invocations distinguished by plugin in terms of the number of processes in the system ( $n$ ), and in the period resolution, that is, the number of time instants the system can distinguish ( $t$ ).

So far we have assumed that the duration of a scheduler invocation is constant or bounded by a constant. However, motivated by our VBS implementation, the

	list	array	matrix/tree
time	$O(n^2)$	$O(\log(t) + n \cdot \log(t))$	$\Theta(t)$
space	$\Theta(n)$	$\Theta(t + n)$	$O(t^2 + n)$

**Table 3.** Time and space complexity per plugin [10, 12]

execution time of the scheduler can vary depending on the number of processes in the system and/or the number of different time instants we can distinguish. This is because the scheduler execution time is composed of the time it takes to manage process queues, which depends on the specific implementation, and the time spent on context switches and other operations that are independent of the implementation. We can thus write that  $\xi = \xi_q + \xi_c$ , where  $\xi_q : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  is a function representing the overhead of managing the queues and  $\xi_c$  is a constant overhead induced by the scheduling algorithm.

## 8 Conclusions

We have introduced response and utilization accounting of scheduler overhead in the schedulability analysis of VBS. Response accounting maintains CPU utilization at the expense of increased response-time bounds whereas utilization accounting maintains the bounds at the expense of increased utilization. Our analysis improves when accounting for scheduler overhead due to releasing processes in a separate, virtual VBS process. We have also shown that our method can readily be generalized to account for non-constant scheduler overhead. Combined response and utilization accounting is possible and may be used in future work to trade-off response time and utilization in solutions to finding appropriate server configurations that also consider scheduler overhead (server design problem).

## References

1. L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.
2. L. Abeni, G. Lipari, and G. Buttazzo. Constant bandwidth vs. proportional share resource allocation. In *Proc. ICMCS*, volume 2, pages 107–111. IEEE Computer Society, 1999.
3. G. Beccari, M. Reggiani, and F. Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Proc. ECRTS*, pages 21–28. IEEE Computer Society, 1999.
4. S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proc. RTSS*, pages 396–408. IEEE Computer Society, 2003.
5. A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed-priority schedulers. *IEEE Transactions on Software Engineering*, 21(5):475–480, 1995.

6. G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.
7. G. Buttazzo. Rate monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29(1):5–26, 2005.
8. G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-Time Systems*, 23(1-2):7–24, 2002.
9. G. Buttazzo and E. Bini. Optimal dimensioning of a constant bandwidth server. In *Proc. RTSS*, pages 169–177. IEEE Computer Society, 2006.
10. S.S. Craciunas, C.M. Kirsch, H. Payer, H. Röck, and A. Sokolova. Programmable temporal isolation through variable-bandwidth servers. In *Proc. SIES*. IEEE Computer Society, 2009.
11. S.S. Craciunas, C.M. Kirsch, H. Payer, H. Röck, A. Sokolova, H. Stadler, and R. Staudinger. The Tiptoe system. <http://tiptoe.cs.uni-salzburg.at>, 2007.
12. S.S. Craciunas, C.M. Kirsch, H. Röck, and A. Sokolova. Real-time scheduling for workload-oriented programming. Technical Report 2008-02, University of Salzburg, September 2008.
13. A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. SIGCOMM*, pages 1–12. ACM, 1989.
14. R. Dobrin and G. Fohler. Reducing the number of preemptions in fixed-priority scheduling. In *Proc. ECRTS*, pages 144–152. IEEE Computer Society, 2004.
15. J. Echague, I. Ripoll, and A. Crespo. Hard real-time preemptively scheduling with high context switch cost. In *Proc. ECRTS*, pages 184–190. IEEE Computer Society, 1995.
16. R. Gopalakrishnan and Gurudatta M. Parulkar. Bringing real-time scheduling theory and practice closer for multimedia computing. In *Proc. SIGMETRICS*, pages 1–12. ACM, 1996.
17. K. Jeffay and D. Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Proc. RTSS*, pages 212–221. IEEE Computer Society, 1993.
18. D.I. Katcher, H. Arakawa, and J.K. Strosnider. Engineering and analysis of fixed-priority schedulers. *IEEE Transactions on Software Engineering*, 19(9):920–934, 1993.
19. C-G Lee, J Hahn, Y-M Seo, S L Min, R Ha, S Hong, C Y Park, M Lee, and C S Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, 47(6):700–713, 1998.
20. T. Nakajima. Resource reservation for adaptive QoS mapping in Real-Time Mach. In *Proc. WPDRTS*. IEEE Computer Society, 1998.
21. A K. Parekh and R G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.
22. R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proc. MMCN*, pages 150–164. ACM, 1998.
23. H. Ramaprasad and F. Mueller. Bounding preemption delay within data cache reference patterns for real-time tasks. In *Proc. RTAS*, pages 71–80. IEEE Computer Society, 2006.
24. I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. RTSS*, pages 2–14. IEEE Computer Society, 2003.
25. M. A. C. Simoes, G. Lima, and E. Camponogara. A GA-based approach to dynamic reconfiguration of real-time systems. In *Proc. APRES*, pages 1–4. IEEE Computer Society, 2008.