Characterizing Variable Task Releases and Processor Capacities

TIK-Report Nr. 45
Updated June 1999

Martin Naedele
Lothar Thiele
Michael Eisenring*

Abstract

For the schedulability analysis of hard real-time systems it is important to use task and resource models that appropriately capture the particular characteristics of the underlying system. In this paper we suggest new task and processor models, the *Variable Task Model* and *Variable Processor Model* , respectively. The new task model is most appropriate for tasks with bursty release characteristics, and the processor model allows to take into account dead time and variable processing speed.

These models are inspired by results from the networking domain [5], but our presentation emphasizes the dual nature of arrival and service curves. Some new properties of service curves are proven.

As an application, it is shown how worst case bounds on task response time can be calculated. A general schedulability test is derived and shown to be equivalent to the tests of Harter [4] and Lehoczky et al. [7] for the case of periodic tasks and constant execution progress.

*ETH Zurich, Computer Engineering and Networks Laboratory, CH-8092 Zurich, email: {naedele,thiele,eisenring}@tik.ee.ethz.ch

Contents

1 Introduction **3**
1.1 Previous task models . . . 3
1.2 Overview . . . 4

2 Terminology **4**

3 Characterization of variable task releases **6**
3.1 Request curve . . . 6
3.2 Constructing the request curve . . . 7
3.2.1 Periodic task . . . 7
3.2.2 Multiframe task . . . 8
3.2.3 General real-time task . . . 8
3.2.4 Tasks specified by utilization bounds . . . 9
3.2.5 Family of periodic tasks . . . 9

4 Characterization of variable processor capacity **9**
4.1 Delivery curve . . . 9
4.2 Tightness of the delivery curve . . . 11

5 Task latency and schedulability **15**
5.1 Combining tasks . . . 15
5.2 Latency . . . 17
5.3 Schedulability test . . . 18

6 Conclusion **20**

A Mathematics **21**
A.1 Min-plus convolution . . . 21
A.2 Min-minus operator . . . 21
A.3 Max-plus convolution . . . 22
A.4 Max-minus operator . . . 22
A.5 Subadditivity . . . 22
A.6 Superadditivity . . . 23

References **25**

1 Introduction

One of the most important design objectives for hard real-time systems is to obtain a high processor utilisation, while still guaranteeing that all hard deadlines are met. Highly utilizing schedules are only possible, if as much knowledge as possible about the task set and the processor characteristics can be taken into account, while preserving the analyzability.

1.1 Previous task models

Tasks are traditionally divided into periodic and aperiodic tasks. The release of a periodic task repeats with a fixed period, whereas for the aperiodic task a minimum interarrival time may be specified. Each release of a task is considered to request the worst case amount of computation capacity from the processor. This model of periodic tasks is the basis for rate-monotonic analysis [8, 10] and its extensions to accommodate aperiodic tasks e. g. [11]. However, there have been several extensions of this coarse classification in order to capture more detailed a-priori knowledge about a certain task: The Hyperperiodic Task Model [1] allows a task to be executed more frequently than its maximum interarrival time prescribes, if the necessary processing capacity is available. The Multiframe Task Model of Mok and Chen [9] describes situations where regularly every n-th release of a task is known to have considerably higher worst case computation demands than the others. For this particular situation Mok and Chen derived a schedulability test that allows a higher utilisation than the well-known test of [8].

The *Variable Task Model* is a general model that contains all those mentioned above. In addition, it is particularly well suited to describe bursty tasks. A bursty task is a task for which several releases may follow upon each other with a short minimum interarrival time T', but of which it is known that only a certain maximum number of releases will occur within a certain longer period. An example for such a bursty task in a satellite control system is given in [2]. Further applications can easily be found in distributed control and multimedia systems. In contrast to the sporadic periodic tasks of [2], in the *Variable Task Model* the minimum allowed interarrival time need not be fixed, but can be stated in terms of utilisation bounds.

This paper also introduces a processor model which allows to express variable processing speed as consequence of background processing as well as dead-time caused by the execution of higher priority tasks, in contrast to the ususal policy of making execution process directly proportional to time progress. It therefore lends itself naturally to the description of preemptive scheduling.

The following results are described in the paper:

- A new task model is introduced which generalizes previous models and is particularly well suited to describe bursty tasks, tasks with variable computation demands, and thus also whole task families.
- A new processor model is introduced which models time-variable processing capabilities.
- Both models are combined and latency bounds as well a general schedulability test are derived.

1.2 Overview

The paper is structured as follows: Section 2 introduces the notation that will be used in later sections and an example which highlights the need for the suggested task model, which is then defined in Section 3. In Section 4 the new processor model is explained, before in Section 5 both task and processor model are combined to calculate the task latency bound and derive a general schedulability test for preemptive fixed priority scheduling. Section 6 concludes the paper. Some mathematical concepts that are used throughout the paper are defined in Appendix A.

2 Terminology

This section defines the terminology used and introduces an example which will be extended in later parts of the paper.

delivery rate s The nonnegative delivery rate $s(t)$ is a measure for the rate with which task execution progresses, e. g. instructions per second.

request rate r The request rate $r(t) \geq 0$ is a train of impulses scaled with the amounts of computation, e, requested by the respective task releases.

accumulated request function R The accumulated request function $R(t) = \int_0^t r(\tau)\, d\tau,\ R(0) = 0,\ R(t) \geq 0$, is a wide-sense increasing function that represents the total amount of computation that has been requested (by a task or from a processor) up to time t.

accumulated capacity function C The accumulated capacity function $C(t) = \int_0^t s(\tau) d\tau,\ C(0) = 0,\ C(t) \geq 0$ is a wide-sense increasing function that represents the total amount of computation the processor could deliver up to time t if it is running at full load.

accumulated computation function R′ The accumulated computation function $R'(t) \geq 0,\ R'(0) = 0$, is a wide-sense increasing function that represents the total amount of computation delivered (from a processor or to a task) up to time t as result of one particular accumulated request function R.

accumulated unused capacity function C′ The accumulated unused capacity function $C'(t) \geq 0,\ C'(0) = 0$, is a wide-sense increasing function that represents the total amount of computational capacity that was not used up to time t, as a consequence of the processor not running at full load.

Progress on task execution in dependence of the available computation capacity and the actual load is calculated using the following equation:

Corollary 2.1 (Relationship between C, R and R′) *For any processor with accumulated request function R(t) and accumulated capacity function C(t) it holds that:*

$$R'(t) = \min_{0 \leq u \leq t} \{R(u) + (C(t) - C(u))\}. \tag{1}$$

Proof: The actually delivered amount of computation, $R'(t)$, must be limited from above by both the accumulated capacity function C(t) and the accumulated request function R(t). In particular, $R'(t)$ is clearly equal to the maximum amount of computation that could have been delivered by t, C(t), minus the maximum wasted capacity up to t. Capacity is wasted during all intervals when R(t) is smaller than C(t). Thus

$$R'(t) = C(t) - \max_{0 \le u \le t} \{C(u) - R(u)\}.$$

$$\iff R'(t) = C(t) + \min_{0 \le u \le t} \{R(u) - C(u)\}$$

$$\iff R'(t) = \min_{0 \le u \le t} \{R(u) + C(t) - C(u)\}.$$

■

The unused capacity can then be calculated with the following corollary:

Corollary 2.2 (Relationship between C, R and C′) *For any processor with accumulated request function R(t) and accumulated capacity function C(t) it holds that:*

$$C'(t) = C(t) - R'(t). \tag{2}$$

Proof: The proof is clear from the definition of C and Corollary 2.1. ■

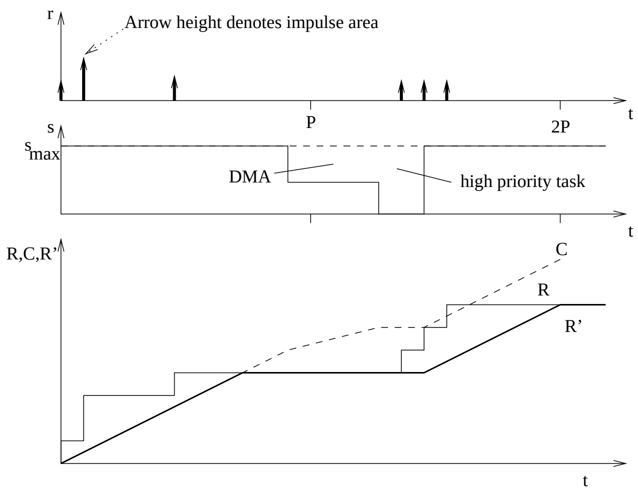

Figure 1: Possible curves for s, r, C, R, and R' for the example in the text.

Example 2.3 *Consider a task τ that processes messages, for example sensor inputs, that arrive via a network. The sensors' transmission instants are not synchronized. The message size varies between a known minimum and maximum. Each message arrival results in the release of a task with worst-case computation demand e', $e'_{min} \le e' \le e'_{max}$, depending on the message size. The minimum interarrival time for messages is T', $T'_{min} \le T' \le T'_{max}$, caused by the fact that the sensor messages have to use the bus and pass through the bus interface in sequence. Suppose that T'*

and c' are related by $T' = u_{var} * (e' - e'_{min}) + T'_{min}$, with the utilization bound for the variable part of the task specification defined as $u_{var} = \frac{T'_{max} - T'_{min}}{e'_{max} - e'_{min}}$, because after a long message it will take longer until the next message is received. Suppose that for this particular application it is also known, that there is a maximum of n=3 message arrivals per interval P ($P > n * T'_{max}$), as in the example of [2]. In certain time intervals, τ is preempted by higher priority tasks, and in other intervals, the execution of τ is not progressing at maximum speed, e. g. because of cycle-stealing DMA.

Fig. 1 shows possible curves s, r, C, R, and R' for this example. The next section describes how to formally characterize such a task set in order to solve scheduling problems.

3 Characterization of variable task releases

3.1 Request curve

Mok and Chen [9] introduce the notion of a General Real-Time Task which is characterized by a minimum distance between its releases and a sequence whose i-th element represents the worst case accumulated computation time for any i consecutive requests. This approach can be taken one step further by adapting the concept of arrival curves from the networking domain [5] for task releases. This is formally expressed in the following definition:

Definition 3.1 (Request curve) *The request curve α of a task with an accumulated request function R is a wide-sense increasing, non-negative function. R is said to be constrained by α, iff*

$$R(t) - R(s) \leq \alpha(t - s) \quad \forall s \leq t. \tag{3}$$

Remark: This definition is equivalent to the statement $R \leq R \underline{\oplus} \alpha$ ($R = R \underline{\oplus} \alpha$ for $\alpha(0) = 0$), which in turn is equivalent to $R \ominus R \leq \alpha$ [5].

If $\alpha(t) = u \cdot t + e$, $e, u = const$, then the task arrival is said to be linearly constrained with parameters (e,u). In this case, e is the upper limit of the worst case amount of computation required, and u represents the maximum utilisation of this task.

The intuitive meaning of a request curve α for a task τ is, that within any time interval of length Δ only a total amount of computation $\alpha(\Delta)$ may be requested by the releases of τ, either through frequent requests with small computational demands, rare requests with high computational demands, or any suitable combination. A task requires the maximum possible computation capacity, if its accumulated request function R equals its arrival curve α.

Definition 3.2 (Greedy task) *A task for which holds $R(t) = \alpha(t)$ is called a greedy task.*

For all practical situations only a subadditive[1] request curve makes sense. If a task is constrained by a request curve α which is not subadditive then it is also constrained by the subadditive request curve $\overline{\alpha}$ (subadditive closure of α). Given an accumulated request function in the t-domain, the request curve in the Δ-domain can be constructed as follows:

[1] Refer to Appendix A for mathematical definitions used in this paper.

Corollary 3.3 (Construction of the minimum request curve) *For a task characterised by an accumulated request function R, the minimum constraining request curve α_{min} (that means: task arrival is constrained by α_{min}, α_{min} is subadditive, and $\alpha_{min} \leq \alpha$) can be calculated as*

$$\alpha_{min}(\Delta) = (R\overline{\ominus}R)(\Delta) = \tag{4}$$

(proposition 4 in [5]).

Example 3.4 *Consider again the example given at the end of Section 2. For simplicity assume that $T'_{min} = 0$ and $e'_{min} = 0$. Fig. 2a) shows a possible curve for the accumulated request function R and how the request curve is constructed from it. Using Definition 3.1 one can easily convince oneself that α in Fig. 2b) corresponds to the minimum request curve to accommodate all possible accumulated request functions that follow the example specification. For comparison there is also a curve R_{wc} which is the worst case bound on all curves R for conventional task models that do not contain an utilization bound but use only minimum interarrival time and maximum computation demand as task characteristics. The request curve α is clearly advantageous for schedulability considerations.*

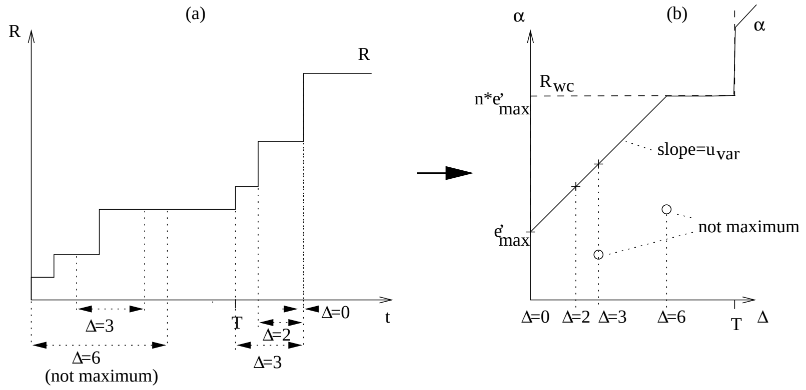

Figure 2: Construction of the request curve .

3.2 Constructing the request curve

The construction of request curves using Corollary 3.3 is computationally expensive. The following subsections give methods that can be used to determine α_{min} for several common special cases.

3.2.1 Periodic task

For a task that is specified by a constant period P and a constant worst-case execution time e, the request curve α is identical to the worst case accumulated request function R, that is, a staircase function with step hight e, first step at time $t = 0^+$ and all consecutive steps as early as possible, at $t = (k * P)^+$, k=1,2,3,4.

3.2.2 Multiframe task

A multiframe task [9] consists of N consecutive requests with, possibly different, worst-case execution times $e_0 \dots e_N$ and minimum separation time P between requests. The whole request pattern repeats cyclically with a hyperperiod of $N \cdot P$.

The request curve can be constructed with the following algorithms, which are equivalent to Corollary 3.3 but make use of the fact that there is only a finite number of time intervals Δ which need to be checked. First a table with the N relevant values for one hyperperiod needs to be calculated:

```
FUNCTION makeTable(INTEGER N, REAL e[]): REAL[]
  // calculate table for all deltas within the first hyperperiod
  INTEGER delta,i,j,
  REAL sum,max,table[]

  table[0]:=0
  FOR delta=0 TO N-1
    max:=0.0
    // try all starting points for a sequence
    FOR i=0 TO N-1
      // sum all sequences of length delta
      sum:=0.0
      FOR j=0 TO delta
         sum:=sum+e[(i+j) mod N]
      END
      IF sum>max THEN max:=sum
    END
    table[delta+1]:=max
  END
  RETURN table[]
END
```

This algorithm is of complexity O(N^3). Exploiting the fact that the periodicity of $N \cdot P$ results in an equal periodicity of the request curve, α is then calculated as follows:

```
FUNCTION alpha(REAL delta,REAL[] table, INTEGER P, INTEGER N) : REAL
  RETURN table[CEILING((delta mod NP)/P)]
END
```

For a multiframe task with the AM-property [9], this reduces to the problem of finding the peak frame of the task. The request curve α is then identical to the accumulated request function R that results from releasing the peak frame at $t = 0^+$ and all other frames in the sequence determined by the task specification, with distance P.

3.2.3 General real-time task

The General Real Time Task [9] is defined by a minimum period P between consecutive releases and a function $\phi(i)$ which returns the maximum sum of the execution times of i consecutive releases of the task. This function is thus equivalent to the table that had to be calculated from the specification for the multiframe task in the previous section. The request curve of a general real time task is thus calculated as $\alpha(\Delta) = \phi(\lceil \frac{\Delta \mod NP}{P} \rceil)$.

3.2.4 Tasks specified by utilization bounds

If a task is specified by a worst-case execution time e and a maximum utilization bound u then the request curve immediately follows as $\alpha(\Delta) = e + u \cdot \Delta$.

3.2.5 Family of periodic tasks

The request curve for a task family is computed by taking the maximum of the request curves of all member tasks for each interval Δ.

4 Characterization of variable processor capacity

4.1 Delivery curve

Usually processors serving task requests are considered to be able to deliver a constant amount of computation per unit time. This is, for example, the implicit basic assumption for the well-known results of Liu and Layland [8] as well as the feasability tests of Harter [4] and Lehoczky et al. [7].

The more general concept of a delivery curve in the *Variable Processor Model* which is inspired by the use of service curves in the networking domain in [5], characterizes a processor via the minimum amount of computation that is guaranteed to be delivered in an interval of length Δ starting at any arbitary point in time t. This leads to the following definition:

Definition 4.1 (Delivery curve) *The delivery curve β of a processor with accumulated capacity function C is a wide-sense increasing, non-negative function. The processor is said to offer a delivery curve of β, iff*

$$C(t) - C(s) \geq \beta(t-s) \, \forall \, t, s : \, s \leq t. \tag{5}$$

The value of the delivery curve $\beta(\Delta)$ for any interval of length Δ can be obtained a-posteriori from a record of the actually available instantaneous delivery rate s(t) (see Fig. 3), by looking at a moving window of size Δ. $\beta(\Delta)$ is then the minimum over all these windows of the integral over the available capacity:

Corollary 4.2 (Construction of the maximum delivery curve) *For a processor characterised by an accumulated service function C, the maximum constraining service curve β_{max} is calculated as*

$$\beta_{max}(\Delta) = (C \underline{\ominus} C)(\Delta) = \tag{6}$$

Proof: It has to be shown that any service curve obtained from equation 6 also fulfills the definition of β, equation 5:

$$\beta_{max}(\Delta) = (C \underline{\ominus} C)(\Delta) = \min_{u \geq 0} \{C(\Delta + u) - C(u)\}$$

$$\Rightarrow \; \forall \, \Delta \, \exists \, u_0 \, : \, \beta_{max}(\Delta) = C(\Delta + u_0) - C(u_0)$$

$$\Rightarrow \forall \Delta : \beta_{max}(\Delta) \leq C(\Delta + u) - C(u) \ \forall u$$

For $t = \Delta + u$, $s = u$ follows

$$C(t) - C(s) \geq \beta_{max}(t - s), \ \forall t, s$$

■

Example 4.3 *Fig. 3 illustrates the construction of the delivery curve from the information given about the delivery rate in Fig. 1. A release of task τ can happen in any of the three different situations of processor availability. When constructing a schedule for hard real-time systems, it is necessary to take the worst case of processor availability into account. This worst-case scenario for the available accumulated computation capacity after a release of τ is captured in the delivery curve β.*

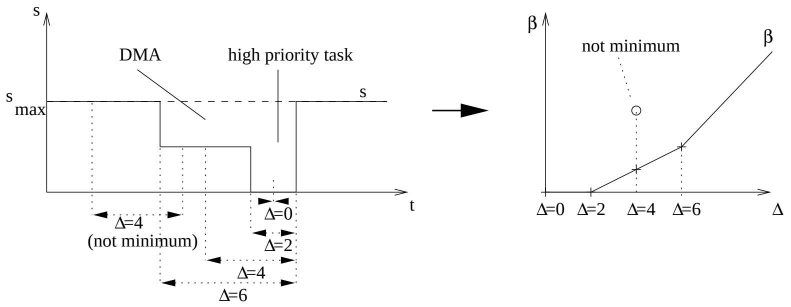

Figure 3: Construction of the delivery curve .

For a processor characterized by a delivery curve β, a lower bound on the actually delivered computation can be given:

Corollary 4.4 (Lower bound on R$'$) *Given is the delivery curve β of a processor. If $\beta(0) = 0$ then for all t there exists a t_0, $0 \leq t_0 \leq t$ such that*

$$R'(t) - R(t_0) \geq \beta(t - t_0), \tag{7}$$

which can also be expressed using min-plus-convolution:

$$R'(t) \geq (R \underline{\oplus} \beta)(t) \tag{8}$$

Proof: Using Corollary 2.1 and Definition 4.1:

$$R'(t) = \min_{0 \leq u \leq t} \{R(u) + (C(t) - C(u))\}.$$

$$R'(t) \geq \min_{0 \leq u \leq t} \{R(u) + \beta(t - u))\} = (R \underline{\oplus} \beta)(t)$$

■

This means, that for a processor characterized by a delivery curve β the allocation of a certain amount of computation capacity can be guaranteed for a task with requests according to a certain accumulated request function R.

Example 4.5 *Fig. 4 illustrates this lower bound: The accumulated request function R shown here is caused by three task releases, and the processor is characterized by the delivery curve constructed above. The accumulated computation function R' is an example for a possible actual task execution progress, as the worst case need not always occur. However, one can easily convince oneself, that for each release of task τ the curve of $R \underline{\oplus} \beta$ is a bound for the respective worst case situation.*

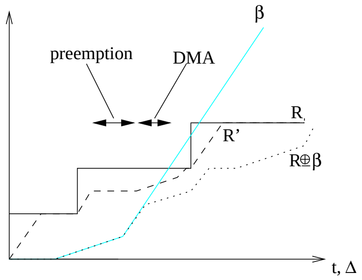

Figure 4: Illustration of the lower bound for R'.

4.2 Tightness of the delivery curve

The following corollaries deal with the question whether and under what conditions this bound is tight.

Corollary 4.6 (Not a tight bound for non-superadditive β) *Be β a service curve such that*

$$\exists\, u, v \ : \ \beta(u+v) < \beta(u) + \beta(v)$$

then $\exists\, R, t$ such that[2]

$$not\ \exists\, C \ : \ (R \underline{\oplus} \beta)(t) - (R \underline{\oplus} \beta)(t_0) = C(t) - C(t_0+)$$

with

$$t_0 = \max\{T : \ 0 \leq T \leq t \ \wedge \ (R \underline{\oplus} \beta)(T) = R(T)\}\,.$$

Proof: Proof is by contradiction. Assume

$$(R \underline{\oplus} \beta)(t) - (R \underline{\oplus} \beta)(t_0) = C(t) - C(t_0+).$$

Remark: Because β is continuous and C is assumed continuous in $]t_0, t+\epsilon]$, $\epsilon > 0$, it holds that $(R \underline{\oplus} \beta)(t_0) = (R \underline{\oplus} \beta)(t_0+)$ and $C(u+v) = C(u+v^+)$.

From the definition of t_0 and with $u = t - t_0$, $v = t_0$ follows that

$$(R \underline{\oplus} \beta)(u+v) - R(v) = C(u+v^+) - C(v^+).$$

Also from Definition A.1:

$$(R \underline{\oplus} \beta)(u+v) \leq \beta(u+v)$$

[2] $C(t_0^+)$ is a short form for $\lim_{h \to 0}(C(t_0+h))$. This notation is also used in later parts of the paper.

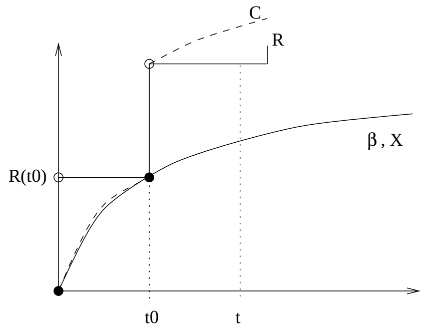

Figure 5: Illustration for the proof to Corollary 4.6

Thus:

$$\beta(u+v) \geq C(u+v^+) - C(v^+) + R(v)$$

Using the precondition:

$$\beta(u) + \beta(v) > \beta(u+v) \geq C(u+v^+) - C(v^+) + R(v)$$

R was chosen such that $R(v) = \beta(v)$ (Fig. 5, gives an example how this could be done), thus

$$\beta(u) + \beta(v) > C(u+v^+) - C(v^+) + \beta(v) \iff \beta(u) > C(u+v^+) - C(v^+),$$

which is a contradiction to the Definiton 4.1 for valid service curves. ■

Corollary 4.7 (Tight bound for superadditive β) *Be β a service curve such that*

$$\forall\, u, v \;:\; \beta(u+v) \geq \beta(u) + \beta(v)$$

then $\exists\, R, t$ such that

$$\exists\, C \,:\, (R \underline{\oplus} \beta)(t) - (R \underline{\oplus} \beta)(t_0) = C(t) - C(t_0^+)$$

with

$$t_0 = \max\left\{T : 0 \leq T \leq t \;\wedge\; (R \underline{\oplus} \beta)(T) = R(T)\right\}.$$

Remark: Because of β continuous, it holds that $(R \underline{\oplus} \beta)(t_0) = (R \underline{\oplus} \beta)(t_0^+)$.

Proof: It is sufficient to prove the claim in one arbitrary interval I where $R(t) > (R \underline{\oplus} \beta)(t)$, because for those intervals J where $R(t) \leq (R \underline{\oplus} \beta)(t)$, C(t) $(t \in J)$ is not bound from above and it is therefore always possible to choose C(t) such that $C(t) - C(s) \geq \beta(t-s),\ t \in J,\, s \leq t$.

Be $I = [t_0, t_1]$.

First, the condition for C will be simplified:

$$(R \underline{\oplus} \beta)(t) - (R \underline{\oplus} \beta)(t_0) = C(t) - C(t_0^+)$$

Because of the the definition of t_0:

$$\iff (R\underline{\oplus}\beta)(t) = C(t) - C(t_0^+) + R(t_0)$$

$$\iff \min_{u:0\leq u\leq t}\{R(u) + \beta(t-u)\} = C(t) - C(t_0+) + R(t_0)$$

Because of the assumption that $R > (R\underline{\oplus}\beta)$ and R, β nondecreasing with $R > \beta$ in $]t_0, t]$ it holds that minimum within the interval is at the left edge of the interval, that is, for $u = t_0$:

$$\iff R(t_0) + \beta(t-t_0)\} = C(t) - C(t_0^+) + R(t_0)$$

$$\iff \beta(t-t_0)\} = C(t) - C(t_0^+)$$

$$\iff C(t) = C(t_0^+) + \beta(t-t_0). \tag{9}$$

This requirement for C can obviously easily be satified by choosing C appropriately. It remains to prove that the accumulated capacity function C obtained from this condition is valid at all, that is whether it also satisfies Definition 4.1. It is thus necessary to prove that Definition 4.1 is satified in any subinterval of I starting at s, $t_0 \leq s \leq t$, not only, which is obvious from Equation 9, in those subintervals starting at t_0:

It in known from Equation 9 that

$$C(t) = C(t_0^+) + \beta(t-t_0) \tag{10}$$

and from the fact that β is superadditive that

$$\beta(t-t_0) \geq \beta(t-s) + \beta(s-t_0).$$

Thus

$$C(t) \geq C(t_0^+) + \beta(t-s) + \beta(s-t_0)$$

and again using Equation 9

$$\iff C(t) \geq C(s) + \beta(t-s).$$

$$\iff C(t) - C(s) \geq \beta(t-s).$$

which satifies Definition 4.1. ∎

The following corollary states that any actual accumulated capacity function $C(t)$ that can be described by a non-superadditive service curve β is also described by the superadditive service curve $\overline{\beta}$, which is the superadditive closure of β. This implies that all reasonable service curves are superadditive:

Corollary 4.8 (Superadditivity closure of β) *For a processor which is characterized by a non-superadditive service curve β there exists no accumulated capacity function C such that $\overline{\beta}(t) > C(t) \geq \beta(t) \wedge C(t) - C(s) \geq \beta(t-s)\ \forall\ s,t : s \leq t$.*

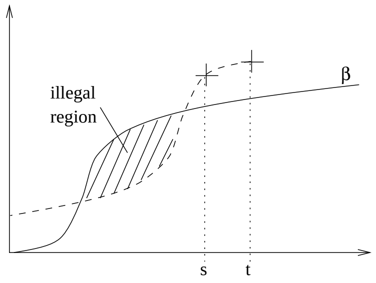

Figure 6: Illustration of the proof for Corollary 4.8

Proof: The corollary is proven by constructing the smallest C that can be described by a given β, and showing that it is identical to the definition of $\overline{\beta}$.

Assume a (non-negative and wide-sense increasing) accumulated capacity function C(t). Choose some arbitray points t and s with $0 < s \leq t$. Be Δ=t-s the distance of those points. From

$$C(t) - C(s) \geq \beta(t-s)$$

it follows that the allowed values of C(s) for the point s are given by

$$C(s) \leq C(t) - \beta(\Delta).$$

See also Fig. 6. On the other hand, it follows from

$$C(s) - C(0) \geq \beta(s-0)$$

that

$$C(s) \geq \beta(s).$$

Thus

$$\beta(s) \leq C(s) \leq C(t) - \beta(t-s) \;\; \forall s \leq t$$

$$=> \beta(s) \leq C(t) - \beta(t-s) \;\; \forall s \leq t$$

$$=> C(t) \geq \beta(s) + \beta(t-s) \;\; \forall s \leq t$$

The goal is to find the minimum C(t), thus

$$C_{min}(t) \geq \max_{0 \leq s \leq t} \{\beta(s) + \beta(t-s)\} = \beta \overline{\oplus} \beta.$$

Now choose a point u<s. As above:

$$C(s) \geq \beta(s)$$

$$C(u) \geq \beta(u)$$

$$C(s) \leq C(t) - \beta(t-s)$$

$$C(u) \leq C(s) - \beta(s-u)$$

and thus

$$\beta(u) \leq C(s) - \beta(s-u) \leq C(t) - \beta(t-s) - \beta(s-u)$$

$$\Rightarrow C(t) \geq \beta(t-s) + \beta(s-u) + \beta(u)$$

and for the minimum:

$$C_{min}(t) \geq \max_{0 \leq u \leq s \leq t} \{\beta(t-s) + \beta(s-u) + \beta(u)\}$$

$$= \max_{0 \leq s \leq t} \left\{ \max_{0 \leq u \leq s} \{\beta(s-u) + \beta(u)\} + \beta(t-s) \right\} = \beta \overline{\oplus} \beta \overline{\oplus} \beta.$$

The above argument can be repeated for all points $v < u$, and so forth. The resulting C_{min} is then given by

$$C_{min}(t) = \max \{\beta, \beta \overline{\oplus} \beta, \beta \overline{\oplus} \beta \overline{\oplus} \beta, \dots\}$$

which is identical to the definition of $\overline{\beta}$. ■

After having found Δ-domain characterizations for both tasks and processors, one can now examine how both fits together and can be used for schedulability analysis.

5 Task latency and schedulability

5.1 Combining tasks

The example for the calculation of the delivery curve (see Fig. 3) already implies, that the delivery curve β, and thus the processing capacity available to a task τ, depends on the raw processing capacity and the capacity that is used by tasks of higher priority than τ. The following theorem formally states how the delivery curve β for all tasks under consideration including τ, the capacity requested by higher priority tasks, α_{prior}, and the delivery curve β' guaranteed to τ, are related:

Theorem 5.1 *Given is a task set Γ of n tasks $\tau_1 \dots \tau_n$, each with an accumulated request function R_i constrained by a request curve α_i. Assume that $\tau_1 \dots \tau_{n-1}$ have higher priority than τ_n. The accumulated request functions R_i, $1 \leq i \leq n-1$ of $\tau_1 \dots \tau_{n-1}$ are constrained by subadditive request curves α_i. The service offered to Γ is characterized by a superadditive delivery curve β. Then the service offered to τ_n is characterized by*

$$\beta'(\Delta) = \max_{0 \leq u \leq \Delta} \{\beta(u) - \alpha_{prior}(u)\} \quad \textit{with} \quad \alpha_{prior} = \sum_{i=1}^{n-1} \alpha_i \tag{11}$$

Proof: From Corrollary 2.2 the actual computation capacity left for task T_n is known to be $C'(t) = C(t) - R'(t)$ with $R'(t) = \sum_{i=1}^{n-1} R'_i(t)$. It is now necessary to prove that β' fulfills Definition 4.1 with respect to C'.

$$C'(t) - C'(s) = C(t) - C(s) -$$

$$-\left(\min_{0\leq u\leq t}\{R(u) + (C(t) - C(u))\} - \min_{0\leq u\leq s}\{R(u) + (C(s) - C(u))\}\right),\ t \geq s$$

C(t) and C(s) are independent of the minimization, thus

$$C'(t) - C'(s) = C(t) - C(s) - C(t) + C(s) -$$

$$-\left(\min_{0\leq u\leq t}\{R(u) - C(u)\} - \min_{0\leq u\leq s}\{R(u) - C(u)\}\right),\ t \geq s$$

$$= \left(\min_{0\leq u\leq s}\{R(u) - C(u)\} - \min_{0\leq u\leq t}\{R(u) - C(u)\}\right),\ t \geq s$$

$$\geq \left(\min_{0\leq u\leq s}\{R(u) - C(u)\} - (R(v) - C(v))\right),\ t \geq s, \forall\, v \leq t$$

$$= \min_{0\leq u\leq s}\{R(u) - C(u) - (R(v) - C(v))\},\ t \geq s, \forall\, v \leq t$$

$$= \min_{0\leq u\leq s}\{(C(v) - C(u)) - (R(v) - R(u))\},\ t \geq s, \forall\, v \leq t$$

$$\geq \min_{0\leq u\leq s}\{\beta(v-u) - \alpha(v-u)\},\ t \geq s, \forall\, v \leq t$$

as this is true for all v, it is also true for the v that causes a maximum:

$$\geq \min_{u\leq s}\left\{\max_{v\leq t}\{\beta(v-u) - \alpha(v-u)\}\right\},\ t \geq s,$$

set d=v-u

$$= \min_{u\leq s}\left\{\max_{d\leq t-u}\{\beta(d) - \alpha(d)\}\right\},\ t \geq s,$$

The minimum of a maximum in a shrinking interval is equal to the maximum in the final (smallest) interval.

$$= \max_{d\leq t-s}\{\beta(d) - \alpha(d)\} = \beta'(t-s),\ t \geq s.$$

■

At the end of this section a comprehensive example (see Fig. 7) will illustrate these concepts.

5.2 Latency

In order to derive schedulability tests, it is necessary to explain how a worst-case bound for the latency of a task can be calculated within the framework of the *Variable Task Model* and the *Variable Processor Model* . For this purpose latency is defined as follows:

Definition 5.2 (Latency) *The latency $L(t)$ is the smallest time delay such that all tasks released at time t have been completed by $t + L(t)$:*

$$L(t) = \min\{T : T \geq 0 \ \wedge \ R(t) \leq R'(t+T)\} \tag{12}$$

To calculate the latency bound, first the concept of the horizontal distance between two curves is introduced:

Definition 5.3 (Horizontal deviation) *For two wide-sense increasing functions α and β the horizontal deviation $h(\alpha, \beta)$ is given by*

$$h(\alpha, \beta) = \max_{s \geq 0} \{\min\{T : T \geq 0 \wedge \alpha(s) \leq \beta(s+T)\}\} \tag{13}$$

The following propertyis valid for h:

Corollary 5.4 *Let f_1 and f_2 be two wide-sense increasing functions with $f_1(0) = f_2(0) = 0$ and $f_1 \leq f_2$. Then for any function f_3*

$$h(f_1, f_2 \oplus f_3) = h(f_1, f_3) \tag{14}$$

The proof is given in [5] (Corollary 1).

The worst-case latency bound for a task can then be shown to be equal to the maximum horizontal distance between α and β (see Fig. 7):

Theorem 5.5 (Latency bound) *For a task constrained by a request curve α, with consecutive releases being served using a FCFS policy, and a processor which guarantees a delivery curve β, the latency $L(t)$ for all t satifies*

$$L(t) \leq h(\alpha, \beta) \ \forall t. \tag{15}$$

Proof: Definition 5.2

$$L(t) = \min\{T : T \geq 0 \ \wedge \ R(t) \leq R'(t+T)\}$$

can also be expressed as

$$L(t) = \max\{T : T \geq 0 \ \wedge \ R(t) > R'(t+T)\}$$

$$\leq \max\{T : T \geq 0 \ \wedge \ R(t) > (R \underline{\oplus} \beta)(t+T)\}$$

$$= \max\left\{T : T \geq 0 \ \wedge \ R(t) > \min_{0 \leq u \leq t+T} \{R(u) + \beta(t+T-u)\}\right\}$$

be u_s the u for which the minimum occurs:

$$= \max\{T : T \geq 0 \ \wedge \ R(t) > R(u_s) + \beta(t+T-u_s)\}.$$

For the case of $u_s \leq t$ use $R(t) - R(s) \leq \alpha(t-s)$

$$L(t) \leq \max\{T : T \geq 0 \ \wedge \ \alpha(t-u_s) + R(u_s) > R(u_s) + \beta(t+T-u_s)\}$$

$$= \max\{T : T \geq 0 \ \wedge \ \alpha(t-u_s) > \beta(t+T-u_s)\}$$

set $v = t - u_s$

$$= \max\{T : T \geq 0 \ \wedge \ \alpha(v) > \beta(v+T)\}$$

As the latency bound is to be valid for all t, it has also to be valid for all $v = t - u_s$, and thus in particular for the v resulting in a maximum T.

$$L(t) \leq \max_{v \geq 0}\{\max\{T : T \geq 0 \ \wedge \ \alpha(v) > \beta(v+T)\}\}$$

$$= \max_{v \geq 0}\{\min\{T : T \geq 0 \ \wedge \ \alpha(v) \leq \beta(v+T)\}\} = h(\alpha, \beta).$$

For the case of $u_s > t$ it is necessary that

$$\min_{0 \leq u \leq t+T}\{R(u) + \beta(t+T-u)\} > R(T)$$

and in consequence the condition to determine T will never be satisfied, so this case can not occur. ■

5.3 Schedulability test

The results of the previous sections are now combined to derive a schedulability test for a task set Γ.

Definition 3.1 and Corollary 3.3 are used to construct the request curve of each task in the task set. Definition 4.1 and Corollary 4.2 define the delivery curve that characterizes the available processing capacity. If the tasks in Γ have different fixed priorities and preemptive scheduling is used, Theorem 5.1 can be used to calculate the delivery curve β_i which is guaranteed to a certain task $\tau_i \in \Gamma$. For the definition of task latency given in Definition 5.2, a worst-case bound on latency is given in Theorem 5.5. From the worst-case bound on latency, the schedulability test follows:

Theorem 5.6 (Schedulability Test) *Assume a processor with delivery curve $\beta(\Delta)$. Then a set of n tasks of decreasing priority which are constrained by request curves $\alpha_i(\Delta)$ and which have relative deadlines D_i is schedulable iff*

$$h(\alpha_i, \beta_i) \leq D_i, \ \forall i : 1 \leq i \leq n \tag{16}$$

with β_i from Theorem 5.1.

The proof is obvious. See Fig. 7 for an example. Clearly the calculations necessary to use this theorem are computationally expensive for arbitary functions α and β. However, the theorem provides a general theory to describe schedulability in the Δ-domain, and for practical cases in which α and β can expressed, either approximated or exactly, as piecewise linear functions, the usual methods can be used to reduce the necessary computational complexity.

Theorem 5.6 can be shown to be equivalent to the well-known schedulability test of Harter [4] and Lehoczky et al. [7] if the appropriate curves for α and β are used:

Corollary 5.7 *Given is a processor with service curve $\beta(\Delta) = \Delta$ and a set of n tasks with increasing periods P_i which are constrained by arrival curves $\alpha_i = e_i \cdot \left\lceil \frac{\Delta}{P_i} \right\rceil$, $e_i \geq 0$. The relative deadlines D_i of the tasks are assumed to be equal to the respective task periods.*

Then the schedulability test

$$h(\alpha_i, \beta_i)) \leq D_i = P_i \tag{17}$$

with

$$\beta_i(\Delta) = \max_{0 \leq u \leq \Delta} \left\{ \beta(u) - \sum_{j=1}^{i-1} \alpha_j(u) \right\} \tag{18}$$

is equivalent to the test of Harter [4] presented in suitable form

$$\min \left\{ T : T \geq 0 \wedge T = c_i + \sum_{j=1}^{i-1} c_j \cdot \left\lceil \frac{T}{T_j} \right\rceil \right\} \leq T_i \tag{19}$$

which is in general equivalent [3] to the test given in [7].

Proof: From [8] it is known that that the maximum for equation 15 occurs for $s = 0^+$, thus equation 13 can here be simplified to

$$\min \left\{ T : T \geq 0 \wedge \alpha_i(0^+) \leq \beta_i(0^+ + T) \right\} \leq P_i$$

$$\Longleftrightarrow \min \left\{ T : T \geq 0 \wedge c_i \leq \max_{0 \leq u \leq T} \left\{ u - \sum_{j=1}^{i-1} c_j \cdot \left\lceil \frac{u}{P_j} \right\rceil \right\} \right\} \leq P_i$$

$$\Longleftrightarrow \min \left\{ T : T \geq 0 \wedge \max_{0 \leq u \leq T} \left\{ u - c_i - \sum_{j=1}^{i-1} c_j \cdot \left\lceil \frac{u}{P_j} \right\rceil \right\} \geq 0 \right\} \leq P_i$$

Because $\beta = \Delta$ is continuous, '$\geq$' can be replaced by '='. The search for T is started with $T = 0$ and thus s=0. As T is increased, a new maximum can occur the first time for an u_t with $u_t = T$. The 'max' can therefore be removed.

$$\min \left\{ T : T \geq 0 \wedge T - c_i - \sum_{j=1}^{i-1} c_j \cdot \left\lceil \frac{T}{P_j} \right\rceil = 0 \right\} \leq P_i$$

This equation is identical to equation 19. ∎

Through suitable modifications of α and β also blocking and release jitter can be represented, which would then allow to derive the iterative general deadline test given in [12].

The schedulability test in the general form of Theorem 5.6 does not make assumptions about the relation between deadlines and task periods. It should however be noted, that for the case of general deadlines it is shown in [6] that neither DMS nor RMS is optimal. See [12] for an algorithm to find an optimal priority assignment.

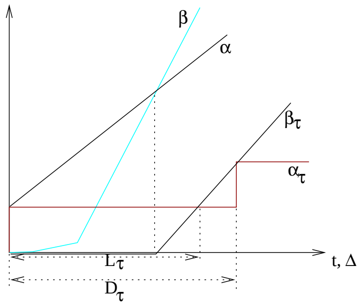

Figure 7: Derivation of β' and schedulability test.

Example 5.8 *Consider again the example and assume that in addition to the message handler a periodic task τ ($\alpha_\tau = R_\tau = e_\tau \cdot \left\lceil \frac{\Delta}{P_\tau} \right\rceil$) of lower priority with $D_\tau = P_\tau$ needs to be scheduled. Fig. 7 shows the previously constructed request curve α (here approximated by an affine function) for the message handler task, the delivery curve β of the processor, and the resulting β_τ left for τ, computed according to Theorem 5.1. As can be seen, the maximum latency L_τ is smaller than the deadline D_τ and the task system is thus schedulable. The figure illustrates that the delivery curve can be interpreted as a bound on deadlines that can be met.*

6 Conclusion

In this paper new models for task releases and computation delivered by a processor were introduced. The *Variable Task Model* is a unified model that incorporates previous models and is, in addition, well suited to describe bursty tasks, tasks with variable computation demands, and task families. The task model does not make any assumptions about deadlines. The *Variable Processor Model* can express dead-time and variable processor speed. It was shown how worst-case latency bounds for a task can be calculated using those models. A general schedulability test was derived from this latency bound.

A Mathematics

A.1 Min-plus convolution

Definition A.1 (Min-plus convolution) *For two functions* γ_1 *and* γ_2 *define* $\gamma_1 \underline{\oplus} \gamma_2$ *by*

$$\gamma_1 \underline{\oplus} \gamma_2(t) = \min_{u: 0 \leq u \leq t} \{\gamma_1(u) + \gamma_2(t-u)\} \tag{20}$$

Min-plus convolution has the following properties [5]:

- $\underline{\oplus}$ is associative and commutative
- If $\gamma_1(0) = \gamma_2(0) = 0$ then $\gamma_1 \underline{\oplus} \gamma_2 \leq min(\gamma_1, \gamma_2)$.
- If γ_1 and γ_2 are concave then $\gamma_1 \underline{\oplus} \gamma_2 = min(\gamma_1, \gamma_2)$.
- In general, the curve of $\gamma_1 \underline{\oplus} \gamma_2$ can be obtained by moving the origin of one of the functions along the curve of the other. The bottom edge of the resulting figure is the curve resulting from the min-plus convolution.
- If γ_1 and γ_2 are convex and piecewise linear then $\gamma_1 \underline{\oplus} \gamma_2$ is obtained by putting the different pieces of both functions end-to-end, sorted by increasing slopes. The piece with the steepest slope is dropped. For an example see Fig. 8.

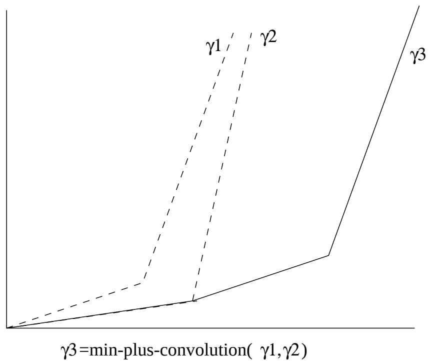

Figure 8: Min-plus convolution for convex, piecewise linear functions

Definition A.2 (Burst delay function)

$$\delta_T(t) = \begin{cases} 0 & : \quad 0 \leq t \leq T \\ +\infty & : \quad T < t \end{cases}$$

Remark: The result of the min-plus convolution of a function $\gamma(t)$ and the burst delay function $\delta_T(t)$ is the original function $\gamma(t)$, shifted T to the right, $(\gamma \underline{\oplus} \delta_T)(t) = \gamma(t-T)$.

A.2 Min-minus operator

Definition A.3 (Min-minus operator) *For two functions* γ_1 *and* γ_2 *define* $\gamma_1 \underline{\ominus} \gamma_2$ *by*

$$\gamma_1 \underline{\ominus} \gamma_2(t) = \min_{u \geq 0} \{\gamma_1(t+u) - \gamma_2(u)\} \tag{21}$$

A.3 Max-plus convolution

Definition A.4 (Max-plus convolution) *For two functions* γ_1 *and* γ_2 *define* $\gamma_1 \overline{\oplus} \gamma_2$ *by*

$$\gamma_1 \overline{\oplus} \gamma_2(t) = \max_{u: 0 \le u \le t} \{\gamma_1(u) + \gamma_2(t-u)\} \tag{22}$$

A.4 Max-minus operator

Definition A.5 (Max-minus operator) *For two functions* γ_1 *and* γ_2 *define* $\gamma_1 \overline{\ominus} \gamma_2$ *by*

$$\gamma_1 \overline{\ominus} \gamma_2(t) = \max_{u \ge 0} \{\gamma_1(t+u) - \gamma_2(u)\} \tag{23}$$

The $\overline{\ominus}$ operator has the following properties ($\gamma_1 \le \gamma_2$ has the meaning $\gamma_1(t) \le \gamma_2(t)$ $\forall t \ge 0$) [5].

- If $\alpha_1 \le \alpha_2$ then for all β: $\alpha_1 \overline{\ominus} \beta \le \alpha_2 \overline{\ominus} \beta$.
- If $\beta_1 \le \beta_2$ then for all α: $\alpha \overline{\ominus} \beta_1 \ge \alpha \overline{\ominus} \beta_2$.
- $(\gamma_1 \overline{\ominus} \gamma_2) \overline{\ominus} \gamma_3 = \gamma_1 \overline{\ominus} (\gamma_2 \underline{\oplus} \gamma_3)$
- $\gamma_1 \overline{\ominus} \gamma_2 \le \gamma_3$ is equivalent to $\gamma_1 \le (\gamma_2 \underline{\oplus} \gamma_3)$

Remark: Intuitively, $(\gamma_1 \overline{\ominus} \gamma_2)(t)$ is the largest difference in value between points on γ_1 and γ_2 that are a distance t apart.

A.5 Subadditivity

Definition A.6 (Subadditivity) *A function* γ *is said to be subadditive if* $\gamma(s+t) \le \gamma(s) + \gamma(t)$ $\forall s, t \ge 0$.

Remark: This definition is equivalent to $\gamma \le \gamma \underline{\oplus} \gamma$. With $\gamma(0) = 0$, subadditivity is equivalent to $\gamma = \gamma \underline{\oplus} \gamma$ [5].

Corollary A.7 *Any concave function* γ *defined on* $\mathbf{R}^+$ *such that* $\gamma(0) = 0$*, is subadditive.*

The proof is given in [5] (Proposition 3).
Remark: A subadditive function γ, even with $\gamma(0) = 0$, is not necessarily concave.

Definition A.8 (Subadditive closure) *The sub-additive closure of a function* γ *is* $\overline{\gamma} = \min_{n \ge 1} \gamma^{(n)}$, *where* $\gamma^{(n)} = \gamma \underline{\oplus} \gamma \underline{\oplus} \ldots \underline{\oplus} \gamma$.

Corollary A.9 *The subadditive closure of a function* γ, $\overline{\gamma}$, *is subadditive.*

The proof is given in [5] (Proposition 2).
Remark: If the function γ is subadditive then $\overline{\gamma} = \gamma$.

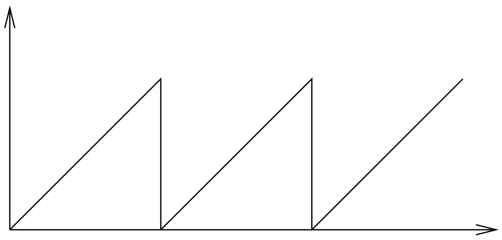

Figure 9: Example of a superadditive function which is not convex and not non-decreasing.

A.6 Superadditivity

Definition A.10 (Superadditivity) *A function γ is said to be superadditive if $\gamma(s+t) \geq \gamma(s) + \gamma(t)\ \forall s, t \geq 0$.*

Remark: A superadditive function f, even with $\gamma(0) = 0$, is not necessarily convex. See Fig. 9

Definition A.11 (Superadditive closure) *The super-additive closure of a function γ is $\overline{\gamma} = \max_{n \geq 1} \gamma^{(n)}$, where $\gamma^{(n)} = \gamma \overline{\oplus} \gamma \overline{\oplus} \ldots \overline{\oplus} \gamma$.*

Corollary A.12 *The superadditive closure of a function γ, $\overline{\gamma}$, is superadditive.*

The proof is analogous to the one for the subadditive closure.
Remark: If the function γ is superadditive then $\overline{\gamma} = \gamma$.

Theorem A.13 *Be β any superadditive and α any subadditive function, then $\gamma = \beta - \alpha$ is also superadditive.*

Proof:

$$\gamma(a+b) = \beta(a+b) - \alpha(a+b)$$

Because of

$$\beta(a+b) \geq \beta(a) + \beta(b) \ \ and \ \ \alpha(a+b) \leq \alpha(a) + \alpha(b)$$

it follows that

$$\gamma(a+b) \geq (\beta(a) + \beta(b)) - (\alpha(a) + \alpha(b))$$

$$= \beta(a) - \alpha(a) + \beta(b) - \alpha(b) = \gamma(a) + \gamma(b) \ \ \forall\, a, b \geq 0$$

■

Theorem A.14 *Be β any superadditive and α any subadditive function, with $\beta, \alpha \geq 0$, and β nondecreasing. If $\alpha(t) < \beta(t) \forall\, t$ then $\gamma = \beta - \alpha$ is also superadditive and nondecreasing.*

Proof: From Theorem A.13 it is known that $\gamma = \beta - \alpha$ is superadditive. It remains to prove that γ is also nondecreasing.

Proof is by contradiction: Assume that there exists an interval I=[t1,t2] where γ is decreasing. As β is nondecreasing, this clearly requires that the slope of α must be steeper than the slope of β in I:

$$\alpha(t_0) - \alpha(t_1) > \beta(t_0) - \beta(t_1),\ t_0 \in I.$$

Using

$$\beta(t_0) \geq \beta(t_1) + \beta(t_0 - t_1)$$

$$\Rightarrow \alpha(t_0) - \alpha(t_1) > \beta(t_1) + \beta(t_0 - t_1) - \beta(t_1) = \beta(t_0 - t_1)$$

Using

$$\alpha(t_1) + \alpha(t_0 - t_1) \geq \alpha(t_0)$$

$$\Rightarrow \alpha(t_1) + \alpha(t_0 - t_1) - \alpha(t_1) = \alpha(t_0 - t_1) > \beta(t_0 - t_1)$$

This is a contradiction to the requirement that $\alpha(t) < \beta(t)\,\forall\, t$. ■

References

[1] S. Aggarwall and C. Chraibi. Hyperperiodic tasks: A new model for hard real-time systems. In *Proceedings IFAC 12th Triennial World Congress*, 1993.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, September 1993.

[3] Neil Audsley, Alan Burns, Robert Davis, Ken Tindell, and Andy Wellings. Fixed priority preemptive scheduling: An historical perspective. *Real-Time Systems*, (8):173–198, 1995.

[4] Paul Harter. Response times in level-structured systems. *ACM Transactions on Computer Systems*, 5(3):232–248, August 1987.

[5] Jean-Yves LeBoudec. Network calculus made easy. Technical Report EPFL-DI 96/218, Ecole Polytechnique Federale, Lausanne (EPFL), 1996.

[6] John Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings IEEE Real-Time Systems Symposium*, 1990.

[7] John Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings IEEE Real-Time Systems Symposium*, 1989.

[8] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), January 1973.

[9] Aloysius Mok and Deji Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–644, October 1997.

[10] Lui Sha, Ragunathan Rajkumar, and Shirish S. Sathaye. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1):68–82, January 1994.

[11] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, (1):27–60, 1989.

[12] K. Tindell, A. Burns, and J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.