A simple approximation method for reducing the complexity of Modular Performance Analysis

Urban Suppiger Simon Perathoner Kai Lampka Lothar Thiele

TIK-Report No. 329

Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH)

August 2010

Abstract

Modular Performance Analysis (MPA) is a method for worst and best case performance analysis of real-time systems. It has been successfully applied to medium scale systems, however, for large scale systems the computation demands increase rapidly. In this paper a simple approximation method is introduced that simplifies the representation of arrival curves, the abstract event stream model at the basis of MPA. The approximation avoids periodic arrival curves and permits to considerably speed up the analysis of large systems, with almost negligible loss of accuracy.

0 Organization

In Section 1 we introduce the basic notions of the Modular Performance Analysis (MPA) and describe how arrival curves are represented in the RTC Toolbox, an implementation of MPA. In Section 2 we propose a simple but efficient approximation scheme for simplifying the representation of arrival curves and hence for reducing the complexity of the performance analysis.

1 Modular Performance Analysis with Real-Time Calculus

Modular Performance Analysis (MPA) [6] is a method for the best and worst case analysis of real time systems. It uses Real-Time Calculus (RTC) [4] as mathematical basis. In this section we shortly introduce the basics of MPA. Detailed explanations have been given by Wandeler et al. [6] and Chakraborty et al. [1]

For practical use of the MPA method a Matlab toolbox, called RTC Toolbox, has been developed [5].

1.1 Basics

The components of distributed embedded systems communicate via message passing. The data message exchange between components can be abstracted by means of event streams. In particular, an input event for a

component represents the arrival of a message, and an output event the sending of a message. In Real-Time Calculus, event streams are abstracted from the time domain and represented in the interval domain by means of so-called arrival curves:

Def. 1. *(Arrival Curve). Let $R[s,\, t)$ denote the number of events that arrive on an event stream in the time interval $[s,\, t)$. Then, the corresponding upper and lower arrival curves are denoted as α^u and α^l, respectively, and satisfy:*

$$\alpha^l(t-s) \leq R[s,\, t) \leq \alpha^u(t-s),\ \ \forall s < t, \tag{1}$$

where $\alpha^u(0) = \alpha^l(0) = 0$.

Characteristics of concrete event streams can be modeled by choosing the arrival curves α^l and α^u in an appropriate way, where α^l belongs to the best case event stream and α^u to the worst case event stream, respectively. Arrival curves are a generalization of standard event models such as periodic or periodic with jitter (PJD) models. In a PJD model the event stream is modeled with a period p, a maximal jitter j with respect to the ideal periodic arrival time and a minimum inter-arrival distance d between any two events. The upper and lower arrival curves α^u and α^l of a PJD event stream model with parameters p, j and d are computed as follows:

$$\alpha^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor,\ \ \alpha^u = \min\left\{ \left\lceil \frac{\Delta + j}{p} \right\rceil,\ \left\lceil \frac{\Delta}{d} \right\rceil \right\} \qquad \Delta > 0 \tag{2}$$

In a similar way, the availability of processing or communication resources is represented by means of service curves:

Def. 2. *(Service Curve). Let $C[s,\, t)$ denote the number of events that a resource can process in the time interval $[s,\, t)$. Then, the corresponding upper and lower service curves are denoted as β^u and β^l, respectively, and satisfy:*

$$\beta^l(t-s) \leq C[s,\, t) \leq \beta^u(t-s),\ \forall s < t, \tag{3}$$

where $\beta^u(0) = \beta^l(0) = 0$.

In RTC α and β must be expressed in the same unit. In particular, we can either express the occurring of events R and the availability of resources C in event units (as done for Def. 1 and Def. 2) or express both R and C in resource units. We denote event-based arrival (service) curves as α (β) and resource based arrival (service) curves with $\tilde{\alpha}$ ($\tilde{\beta}$).

The simplest resource model is a fully available resource which corresponds to a pair of service curves β^l and β^u with slope 1.

As we will describe in detail below, this thesis only focuses on the analysis of communication tasks and neglects computation tasks. Therefore, in the context of this thesis events actually mean messages and resources always denote link bandwidth.

1.2 Curve representation

In the RTC Toolbox, arrival and service curves are internally represented as a concatenation of linear segments. A curve consists of an aperiodic part, followed by a periodic part. Each of these two parts consists of a finite list of segments. Each segment is represented by 3 values (x, y, s), where (x, y) defines the start point of the segment and s defines its slope. The end of a segment is implicitly set by the start point of the next segment. If there is no next segment, the last segment is endless.

The following values represent the curve in Fig. 1:

Listing 1. Example curve representation

```
01 Curve:
02    AperiodicPart   = {(0.0,0.0,1.0)(2.0,2.0,0.5)}
03    PeriodicPart    = {(0.0,0.0,0.0)(3.0,1.0,0.0)}
04     (px0,py0)      = (6.0,7.0)
05     (pdx,pdy)      = (4.0,4.0)
```

AperiodicPart and *PeriodicPart* both define the previously mentioned segment lists. Additionally, the following values are defined.

(*px0*,*py0*) Start point of the periodic part.

(*pdx*,*pdy*) The difference on the x- and y-axis between two consecutive repetitions of the periodic part. *pdx* is also called period.

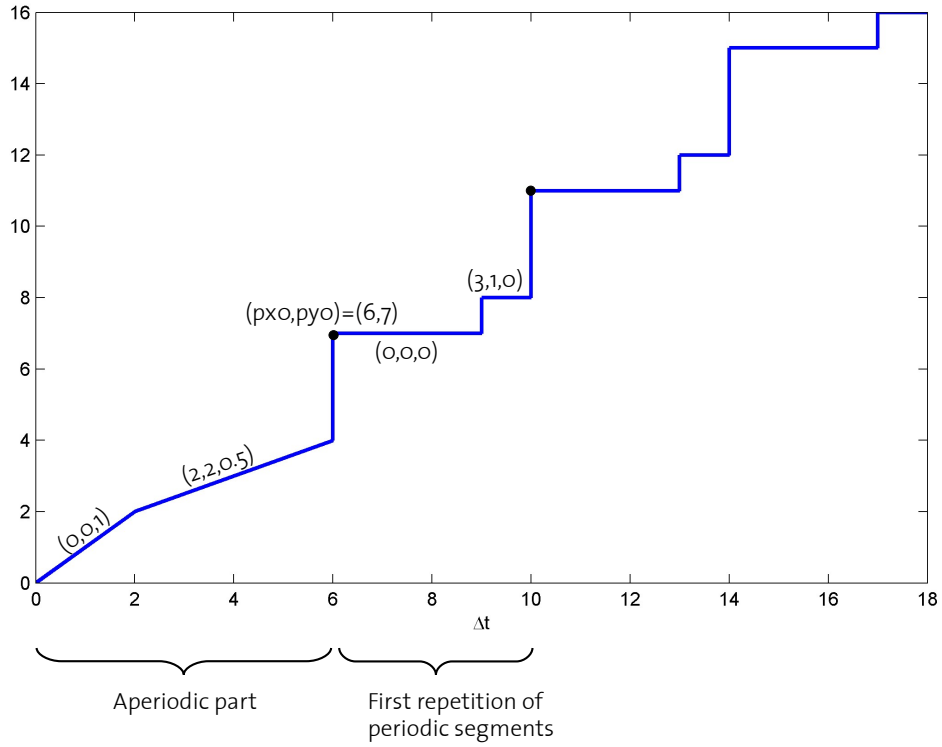

Figure 1. Example of a RTC curve with aperiodic and periodic part

1.3 Standard Components

In real systems, tasks process event streams by using resources. The output of the tasks is an output event stream and a remaining resource availability, which denotes the amount of resources available after the processing of the task. In MPA these tasks are represented as RTC components, which take arrival curves α and a service curve β as inputs and calculate the outgoing arrival curves α' and an outgoing service curve β'. α' represents the output event stream in the interval domain and β' represents the remaining resource availability in the interval domain.

Several processing components are defined in the MPA framework. In the following, we briefly describe the components of MPA that are relevant for this thesis.

1.3.1 Greedy Processing Component (GPC)

The GPC component has a single arrival curve and a single service curve as input. It models the case where only one task is processed on a single resource. The GPC component processes incoming events in a greedy fashion in first-in-first-out order while being bounded by the resource availability described by the service curve β. Many other components use GPC as a basic component.

1.3.2 First In First Out (FIFO)

The FIFO component models the case where multiple tasks are processed on a single resource in a first-in-first-out manner, which means that the events are processed in their incoming order. Because the processing order is already determined by the incoming event stream, no process can be interrupted by another event and so the FIFO component is a non-preemptive component. Accordingly, the FIFO component in MPA has one service curve and several arrival curves as input. The mathematical details have been derived in [3].

1.3.3 Non-Preemptive Fixed Priority (NPFP)

The functionality of a NPFP component is similar to a fixed priority (FP) component, where multiple tasks are processed on a single resource and where the active task with the highest priority receives the full resource. "Fixed" means that the priority is statically allocated to each task and does not change. "Active" tasks are those tasks which have events waiting to be scheduled. However, in NPFP the processing of a task is not allowed to be interrupted. The component has to wait until the end of this task before it decides which task will be scheduled next. This results in lower priority tasks being able to backlog higher priority tasks. The mathematical details for the NPFP component, which we have implemented for the RTC Toolbox, have been derived in [2].

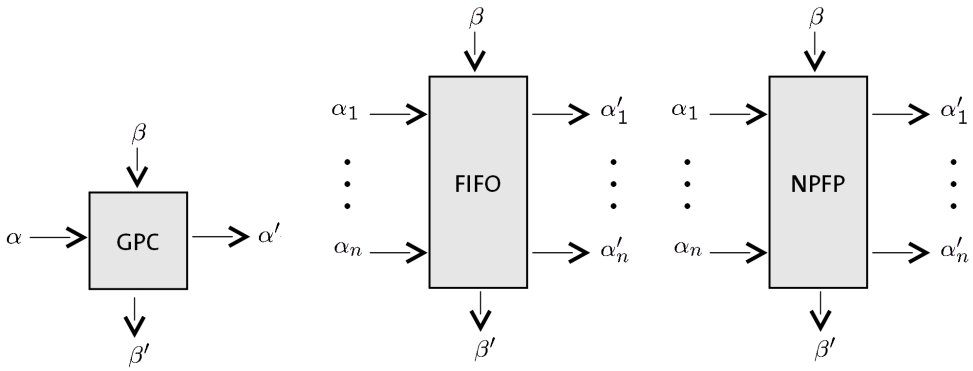

Figure 2. GPC, FIFO and NPFP component

1.4 Analysis

In the Modular Performance Analysis framework, individual components are interconnected to model a larger system. The output arrival curves of a component can be connected to the input of the next component and eventually build a task sequence where data is forwarded from one component to the next. In our case this corresponds to network packets that are forwarded from one device to the next by using network links.

In such systems there are two important performance metrics that can be calculated on system or on component level:

- Worst case buffer requirements

- Worst case end-to-end delay

Below, the calculation of those 2 values for a GPC component is explained. For other components, the calculations may differ slightly, however, the idea remains the same.

Fig. 3 illustrates how worst case delays and buffer requirements are determined in RTC. The worst case delay $d_{\max}$ of a GPC component can be derived by looking at the upper arrival curve α^u and the lower service curve β^l. The maximum horizontal difference between these 2 curves represents the maximum delay an event can experience at this component. It denotes the time between the occurrence of some events and the moment when enough resource is available to process these events.

Similarly, the worst case buffer requirement (or backlog) $b_{\max}$ can be derived by the same curves. The maximum vertical distance between the arrival and the service curve represents the maximum backlog. This is the maximum difference between the number of arrived events and the number of already processed events.

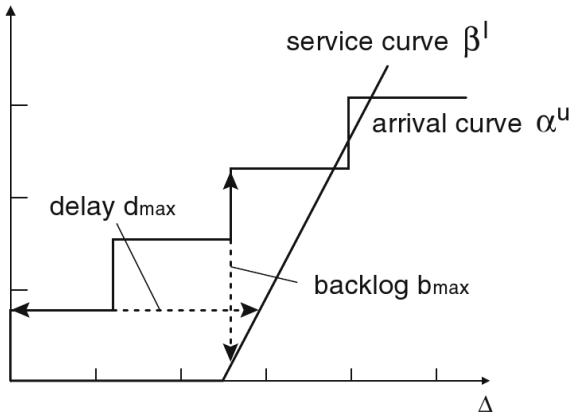

Figure 3. Analysis of $d_{\max}$ and $b_{\max}$

2 Approximation

The RTC Toolbox has been successfully used for the analysis of medium-scale distributed embedded systems. The analysis of large-scale, industrial systems can, however, be problematic for the RTC Toolbox. In particular, the analysis of large systems can lead to very long run-times. In the worst-case, the analysis cannot be completed due to lack of memory. In this section we first try to give reasons for the problems mentioned above. Then we provide approximation methods for simplifying the analysis. The goal is to eliminate the mentioned efficiency problems while preserving the accuracy of the results as much as possible.

2.1 Time and memory consuming calculations

Generally, the more segments that arrival and service curves involved in the operations have, the longer take the operations on these curves and the more memory is consumed. However, there also appear specific problems with periodic curves (curves with periodic parts) and two of these problems will be explained below.

2.1.1 Operations on multiple periodic curves

The operations in the RTC Toolbox heavily affect the curve representations of arrival and service curves. A particular problem is the growth of the periods and the number of segments in the periodic part of curves. Many

RTC operators generate curves with much longer periods than the periods of the corresponding input curves. Examples for such operations are plus, minus or convolution of curves. The period of the output curve often equals to the least common multiplier (LCM) of the input periods. In some seldom cases, the period of the output curve may be smaller than the LCM due to coincidences. However, very often the periodic part of a resulting curve has much more segments compared to the input curves.

In a large system with many components but with a limited number of different curve periods at the sources, one could calculate, that the period of the curve theoretically never exceeds the LCM of all input curves and therefore would stay stable. However, due to cutoff errors when using floating point data types in the RTC Toolbox, this does not hold in practice.

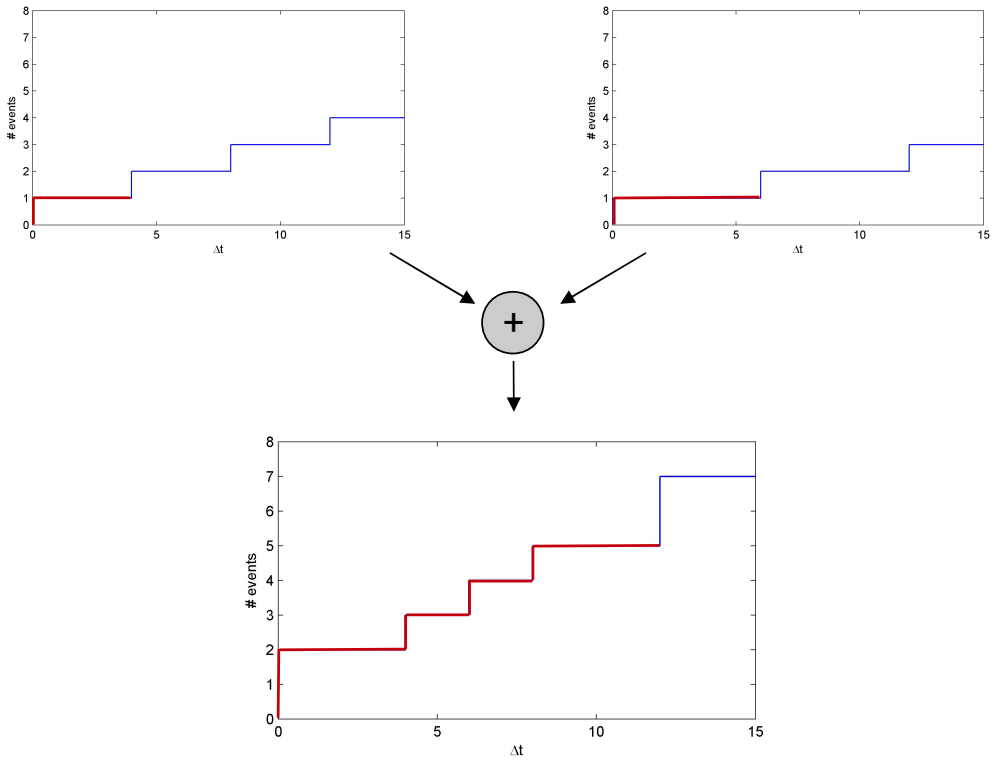

Figure 4. Example of a plus operator on two periodic curves

An example is given in Fig. 4. The example shows the sum calculation of two curves. The first curve is a periodic curve with period 4. The periodic part consists of one single segment.[1] The second curve has a period 6 and again, the periodic part consists of one segment. Applying the plus-operator to these 2 curves, the result is a new periodic curve with a period of 12, which is equal to the least common multiplier of 4 and 6. However, the periodic part of the results now has 4 segments.

[1]The periodic part of the first curve consists of two linear sections, a vertical one and a horizontal one. However, in our curve representation, which was introduced in Section 1.2, only horizontontal and sloped sections are represented as individual segments, whereas vertical sections are represented by choosing the start point of the next segment accordingly. Therefore, we say that the curve consists of one segment only.

2.1.2 Floor and Ceil Operations

Many RTC components apply floor and ceil operations on arrival and service curves. When calculating the floor of a periodic curve with a non integer pdy[2], the curve needs to be unfolded n times such that $n \cdot pdy$ equals an integer value. For example a periodic curve with $pdy = 2.2$ needs to be unfolded $n = 5$ times and the resulting curve has a period 5 times as large as the period of the input curve. Again, cutoff errors in the toolbox implementation can lead to a period increase where none would have been expected. The very same problem also applies to the ceil operator.

Both problems above become worse when several components (or operators) are sequentially placed in a row. Most RTC components consist of many operations, including operations on multiple periodic curves and floor or ceil operations. Components that are connected serially steadily increase the number of segments on the arrival curves and so the runtime of the components heavily increases when stepping forward in the sequence. This is the reason why the analysis of large-scale system becomes problematic.

2.2 Safe approximation

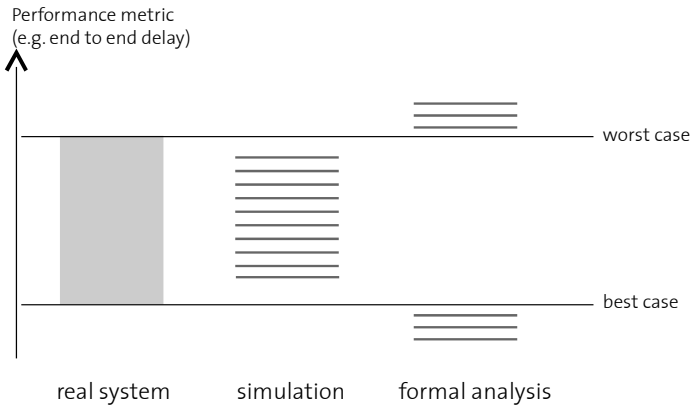

Figure 5. Overview of worst case and best case analysis

In order to prevent large systems to become unanalyzable, we propose to approximate all arrival curves before they enter a RTC component. In this paper we consider only the approximation of arrival curves. However, the discussed ideas can also be applied for the approximation of service curves.

Approximations in a worst case (WC) analysis need to be done in a conservative manner. If we provide the result of a worst case analysis of a system, one can rely on the fact that no behavior of the system is worse than these results. In context of Fig. 5 WC analysis provides an upper bound for the system behavior. Although it is desirable to provide as tight bounds as possible, one could also provide higher WC results that are still correct — they still represent a worst case upper bound. Therefore, an approximated WC analysis is allowed to provide worse results than the exact WC analysis and it may indeed be reasonable if we can conduct it in a more efficient manner. The same thoughts also apply to best case analysis.

In the context of RTC the approximation of an analysis finally breaks down to approximating arrival and service curves, that means to transform them into less complex curves. In order to provide safe approximation of the arrival curves, the values of the approximated upper curve can be higher and the ones of the approximated lower curve can be lower than the exact curves. This leads to the following definition:

[2] pdy represents the offset on the y-axis between two consecutive repetitions of the periodic part.

Def. 3. *(Safe approximation of arrival curves) An approximation $\hat{\alpha}$ of an arrival curve pair $\alpha := [\alpha^u, \alpha^l]$ is called safe, if, and only if,*

$$\hat{\alpha}^l(\Delta) \leq \alpha^l(\Delta), \ \forall \Delta \geq 0 \quad (4)$$

$$\hat{\alpha}^u(\Delta) \geq \alpha^u(\Delta), \ \forall \Delta \geq 0 \quad (5)$$

For the service curves, the following similar definition holds:

Def. 4. *(Safe approximation of service curves) An approximation $\hat{\beta}$ of a service curve pair $\beta := [\beta^u, \beta^l]$ is called safe, if, and only if,*

$$\hat{\beta}^l(\Delta) \leq \beta^l(\Delta), \ \forall \Delta \geq 0 \quad (6)$$

$$\hat{\beta}^u(\Delta) \geq \beta^u(\Delta), \ \forall \Delta \geq 0 \quad (7)$$

2.3 Converting to Aperiodic Curves

In order to prevent the problems mentioned above, we propose an approximation method which limits the number of segments of a curve by avoiding curves with periodic parts. The main idea is, that arrival curves and service curves are defined for all $\Delta \geq 0$, however, usually only intervals of limited size are relevant for the computation of worst case delay and backlog.

Thm. 1. *For any feasible system there is an end bound c_{end} such that when only values at $\Delta < c_{\text{end}}$ are considered, the analysis of the system still results in exact calculations of message delays and buffer requirements.*

Proof (sketch): The proof of this theorem is obvious when Fig. 3 is taken into account. Assumed that $c_{\text{end}} \to \infty$, which means that all values $\Delta \geq 0$ have to be considered in order to retrieve exact results, this implies that either the end point of the maximum horizontal distance goes to infinity or the vertical distance diverges with $\Delta \to \infty$. In both cases, infinite delays or infinite buffer requirements occur and the system is not feasible. □

Based on that, we propose the following approximation method: For each curve an approximation limit c is defined. For all $\Delta < c$, the curve is unfolded and all values are not approximated but represented exactly. Unfolding means that the periodic part is transformed into the aperiodic part by adding the segments of the periodic part to the aperiodic one as often as necessary. All values $\Delta \geq c$ are approximated by a one segment (which could have a vertical or horizontal offset), where the slope of the last segment is equal to the long term rate of the curve. The results is a curve $\hat{\alpha}_c$ with an aperiodic part only. If we choose c such that $c \geq c_{\text{end}}$ the analysis still provides exact results.

Let us assume that we calculate the worst case end-to-end delay d of a system. Choosing $c \geq c_{\text{end}}$ results in a exact worst case delay d_{end}, which defines the lower bound of the worst case calculation. If we choose $c = 0$, which means that we approximate the whole curve by one segment only, the analysis results in an upper bound worst case delay $d_0 > d_{\text{end}}$.

Thm. 2. *For any c_1 and c_2 with $0 \leq c_1 \leq c_2 \leq c_{\text{end}}$ the calculation of the worst case end-to-end delay d_1 and d_2 results in $d_0 \geq d_1 \geq d_2 \geq d_{\text{end}}$.*

Proof (sketch): The main statement of this theorem is, that the results d become exacter (closer to d_{end}) for larger c. The approximated curve set $\hat{\alpha}_{c_1}$ is an outer bound of $\hat{\alpha}_{c_2}$. Outer bound means that the upper curve is higher (or equal) and the lower curve is lower (or equal). In other terms, $\hat{\alpha}_{c_2}$ is a better and tighter approximation of α.

In order to calculate the worst case end-to-end delay, we need to measure the maximum horizontal distance between the upper arrival curve and the lower service curve (see Fig. 3). For $\hat{\alpha}_{c_2}$ the upper arrival curve is lower than for $\hat{\alpha}_{c_1}$ and therefore the horizontal distance to the service curve is smaller for c_2. □

2.3.1 Choosing the right approximation limit c

As we can see, the value of c directly influences the system's performance. A larger approximation limit c leads to more accurate results, however, a large c also leads to more complex curves consisting of a larger number of segments. It seems obvious that this directly results in longer computation times and larger memory consumption.

We propose two different methods to determine the approximation limit. In the first method, c is statically defined by the user for the whole system. In that case, c is an additional design parameter of the model.

In the second method, we adapt c dynamically to each single curve with the following expression:

$$c = pd0 + R \cdot pdx \tag{8}$$

In section 1.2 we introduced *pd0* as the start point of the periodic part, which equals to the length of the aperiodic part, and *pdx* as the period. In the dynamic approximation method, *R* is a design parameter which needs to be chosen by the user and denotes how many repetitions of the periodic part are unfolded and inserted into the aperiodic part. Fig. 6 shows an example.

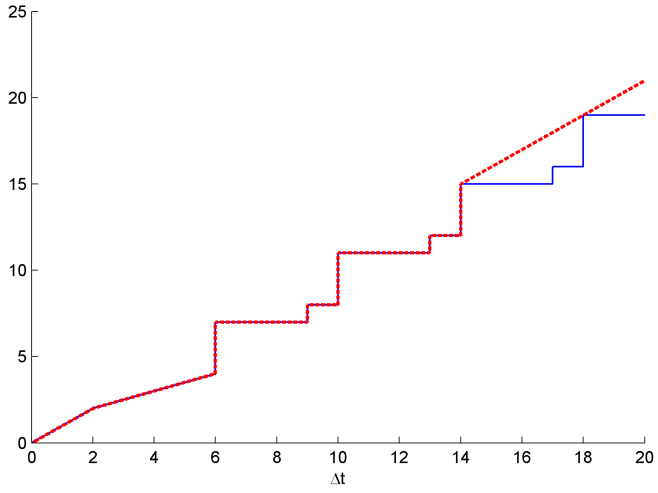

Figure 6. Example of an upper curve approximation (red dashed) with $R = 2$. The aperiodic part of the original curve (blue solid) has length 6 and the periodic part has a period of 4. Therefore, the curve is approximated at $c = 14$ with one linear segment.

3 Conclusions

We have described an approximation method that simplifies the representation of arrival curves, the abstract event stream model at the basis of MPA. The approximation avoids periodic patterns in the representation of arrival curves. The method was successfully applied to an industrial case study. For the considered system, which has over 200 components, the run-time of the analysis could be reduced from 52 seconds to roughly half a second, with almost negligible loss of analysis accuracy. In other words, the described method reduced the analysis effort by two orders of magnitude without affecting the analysis results. The details of the case study are not included in this paper, as they are based on confidential data.

References

[1] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. 6th Design, Automation and Test in Europe (DATE)*, pages 190–195, Munich, Germany, March 2003.

[2] W. Haid and L. Thiele. Complex task activation schemes in system level performance analysis. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 173–178, New York, NY, USA, 2007. ACM.

[3] S. Perathoner, T. Rein, L. Thiele, K. Lampka, and J. Rox. Modeling structured event streams in system level performance analysis. *SIGPLAN Not.*, 45(4):37–46, 2010.

[4] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems ISCAS 2000*, volume 4, pages 101–104, Geneva, Switzerland, 2000.

[5] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[6] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis: a case study. *Int. J. Softw. Tools Technol. Transf.*, 8(6):649–667, 2006.