An Event Stream Driven Approximation for the Analysis of Real-Time Systems*

Karsten Albers, Frank Slomka
Department of Computer Science, University of Oldenburg
Ammerländer Heerstraße 114-118, 26111 Oldenburg, Germany
{karsten.albers,frank.slomka}@informatik.uni-oldenburg.de

Abstract

This paper presents a new approach to understand the event stream model. Additionally a new approximation algorithm for the feasibility test of the sporadic and the generalized multiframe task system scheduled by earliest deadline first is presented. The new algorithm has a polynomial complexity to solve the Co-NP hard problem of schedulability analysis. The approximation error of the algorithm is bounded. In contrary to earlier work, where the error depends on the different deadlines of the tasks, the error of our algorithm depends only on the capacity of the chosen processor. It guarantees the acceptances of a processor with a slightly higher capacity than the unknown optimal processor. While the algorithm is scalable and the run-time depends on the chosen error, a trade-off between running time and error is possible.

1. Introduction

The analysis of the time behavior of embedded real-time systems is a corner stone for the automation of the design process of such systems. To guarantee hard deadlines in real-time systems, schedulability tests have been widely studied in recent years [1], [2], [3], [5], [6], [7], [8], [9], [12], [13]. A good introduction to the area of real-time analysis is given in [5].

Consider the following simple sporadic task model: Given is a set of tasks. Each task is described by

- an initial release time *a*,
- a relative deadline *d* (measured from the release time),
- a worst-case execution time *c* and
- a minimal distance (or period) *p*

between two instances of the task. All tasks are mapped on the same processor (or other hardware units). The tasks are scheduled by the earliest deadline first algorithm (*EDF*). This means the task which has the shortest distance in time to the end of its deadline is executed first.

If the deadlines are not equal to the periods of the tasks, the schedulability test is a Co-NP hard problem [13]. Today no algorithms with polynomial complexity exist. The best known algorithms are algorithms with a pseudo-polynomial complexity.

To use such a schedulability analysis by automatic synthesis tools, it is necessary to reduce the complexity of the test. This can be done by approximation [10] which results in accepting a small error on the final results of the algorithm. A first approach has been the analysis by Baruah et al. [3]. This paper is the first introduction to the processor demand test based on the demand bound function. The main idea of this algorithm is to calculate the complete execution time of all tasks with a start time and a deadline within a given interval *I*. If this amount is not higher than the available execution time of the processor, and if this check is done for all relevant intervals, the schedulability test succeeds.

Baruah proved that this test has only to be performed up to an maximum interval I_{max}. To get a pseudo-polynomial complexity of the algorithm, Baruah considers only such task systems which have an processor utilization below the given bound. This leads to an upper bound of I_{max}.

Spuri presented in [15] a response time analysis for EDF with a pseudo-polynomial complexity [16]. But the algorithm needs more effort than the processor demand test.

The complexity of all these tests are depending not only on the number of tasks, but also on the ratio of the different periods of them. Recently, Chakraborty et al. [6] presented a different approach, which solves this problem and leads to an algorithm with polynomial complexity. The algorithm is defined for an advanced task model, the recurring real-time task model [1], [2]. A fixed amount of test intervals are distributed evenly over

* The research described is supported by the *Deutsche Forschungsgemeinschaft* under grant SL 48/ 1-1

the maximum interval I_{max}. The cumulated execution time is checked only for these intervals. To guarantee the correct behavior of the test also for intervals in between the test intervals, the cumulated execution time is compared against the capacity of the next smaller test interval. The maximum error of this approximation is bounded. It is equal to the distance between two test intervals. If for all instances of all tasks, the distance between the end of their execution and their deadline is larger than the error, the test always succeeds.

Using such an approximation, a trade-off between the error and the execution time of the algorithm is possible. A large error results in only a few test intervals and so in a fast execution, a small error results in many test intervals and a slow execution. If a task system contains tasks with small deadlines, the distance between the end of their executions and their deadline is also short. To assure the acceptance of the task system, the test include deliberately a small error. This leads to a long execution time of the algorithm. The algorithm will always fail with tasks having the same deadline as their worst case execution time.

In this paper a new approach to solve this problem is presented. The new algorithm is independent from the ratio and the size of the deadlines and periods. It uses a different kind of error. The algorithm guarantees the acceptance of a task system running on a processor with a slightly higher capacity than the unknown optimal processor. Such an optimal processor is a processor whose capacity can not be reduced without making the task system infeasible. This error is sizeable and leads to a scalable analysis algorithm. Using this analysis technique it is possible to make a trade-off between the run-time of the algorithm and the size of the error. The correctness of the presented approximation algorithm is formally proven. Additionally, the paper shows a new formal way to derive the event stream approach given by Gresser in [9] from a more general stimuli description. This allows a complete new interpretation of the demand bound function given by Baruah and supports the interpretation of the new approximation algorithm.

2. Model

2.1. Event Stream Model

The event stream model first introduced by Gresser [9] describes the worst case timing relationships between events. The idea is to define a minimal distance in time between one, two, three or more events in a formal specification of the input stimuli. The model defines the maximum number of events in different given time intervals. To formulate the problem in a formal way the following definitions are needed[1]:

1. Equations originally introduced by Gresser are marked with (*)

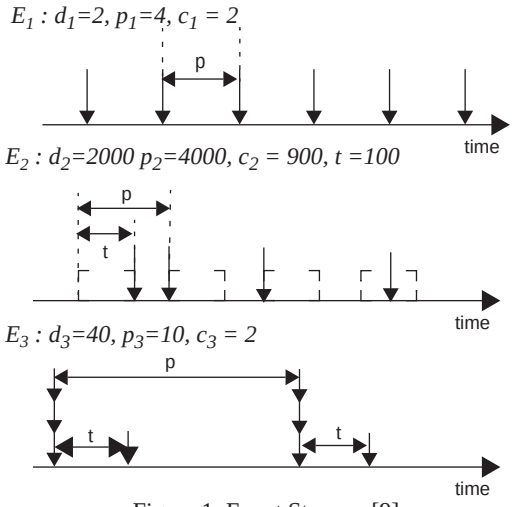

Figure 1: Event Streams [9]

Definition 1: Time T: *The time T is a monotone ordered set of numbers,* $t \in R^+$ *, which define the multiplier of a given physical time unit. Each number is a time point* $t \in T$ *.* Note that each time point t_i can be described by the interval $I(0, t_i)$.

Definition 2:Specific Time Interval:*A specific time interval is a time interval between two specific time points.*

$$I_S = (t_i, t_j)$$

Definition 3: Time Interval I: *A time interval is the distance in time between two time points* t_i *and* t_j*:*

$$I(I_S) = |t_i - t_j|$$

Definition 4: Execution Time c: *The time interval a processor needs to compute a specific piece of software code.*

Definition 5: Task τ: *A task* τ *described by the worst case execution time c and the relative deadline d representing a piece of software code. The relative deadline is an interval I describing the maximum allowed processing time of a task* τ*:*

$$\tau = (c, d)$$

In this paper deadlines are relative hard deadlines. A processor can only compute one task at time. In the following scheduling with the earliest deadline first (EDF) algorithm is assumed.

Definition 6: Event *e*: *An event is a request for the execution of a task at a specific point in time* t_r *(request time):*

$$e = (t_r, \tau)$$

$$t(e) = t_r$$

$$\tau(e) = \tau$$

Definition 7: Event Sequence *E*: *An event sequence is an ordered set of events:*

$$E = \{e \mid (t(e_i) \geq t(e_j) \Leftrightarrow i \geq j)\}$$

Definition 8: Event Interval Function $n_i(E, I)$: *The event interval function defines for an specific time interval*

I the number of events which occur within this time interval:

$$n_i(E, I_S) = \left|\{e \in E | t_i \le t(e) \le t_j\}\right|$$

Definition 9: Homogenous Event Sequence E_H: *A homogenous event sequence is an event sequence consisting solely of events from the same task:*

$$E_H = \{e \in E | \tau(e_i) = \tau(e_j)\}$$

Definition 10: Periodic Event Sequence E_P: *A periodic event sequence is a homogenous event sequence consisting of an infinite amount of events having a fixed distance (the period p). The request time of the first event is the initial request time a of the sequence:*

$$E_P = \left(E_H \middle| \begin{pmatrix} \forall e \in E_H \middle| \exists k \in N_0^+ : (k \cdot p + a = t(e)) \\ \forall k \in N_0^+ \middle| \exists e \in E_H : (k \cdot p + a = t(e)) \end{pmatrix}\right)$$

A periodic event sequence is described by its period *p*, its initial request time *a* and the values from the underlaying task, the worst case execution time *c* and the relative deadline *d* of the corresponding task:

$$E_P = (p_i, a_i, d_i, c_i) \qquad (*)$$

A periodic event sequences with an infinite period is an event sequence consisting of one single event.

Lemma 1: *Each event sequence can be represented by a set of periodic event sequences E_P: $E = \{E_P\}$*

All periodic parts of *E* can be represented by a single periodic event sequence E_P. The others are represented by a periodic event sequence with an infinite period. However, for more general event sequences the number of E_P can become quite large. For most systems only a finite set of E_P's is needed. In the considered sporadic task system, each task can be described with only one E_P and just two E_P's if jitter is taken into account. To define a test algorithm only a upper bound of the density of events must be considered:

Definition 11: Event Function n_e: *The Event function defines for each time interval I the worst case number of events which can occur in I [9]:*

$$n_e(E, I) = max\left\{n \in N_0^+ \middle| \exists t \in T : n = n_i(E, (t, t + I))\right\}$$

The event function is a monotonic non decreasing function. It is complex to extract a description of this function from a given event sequence. For that reason we consider the inverse function:

Definition 12: Event Stream Function *a(n)*: *An event stream function defines for each number n of events the minimum interval I in which n events are able to occur:*

$$a(n) = min\{I \in T | (\exists t \in T : n = n_e(E, I))\}$$

Definition 13: Event Stream E_S: *An event stream is a special case of an event sequence with all events arriving in their worst case timing relationship describing the event stream function.*

$$E_S = \{e \in E | \exists n \in N_0^+ : I(0, t(e)) = a(n)\}$$

$$E_S = \{(p_i, a_i, d_i, c_i)\} \qquad (*)$$

Definition 14: Event Stream Element: *The event stream element is a periodic event sequence which is part of an event stream.*

The worst case timing relationship describes the worst case density of events. An interval *a(n)* is described by the request time interval of an event $e_i \in E_S$. The first event of the event stream represents the marginal interval containing at maximum one event (which is always 0 in the borderline case), the second event represents the interval containing two events and the *n*-th event represents a interval with *n* events.

Example: *The event sequence E_1={0, p_1, 2p_1, 3p_1, ...} of Fig. 1 is a periodic event sequence. Therefor the event stream E_{S1} is given by E_{S1} = {(p_1, 0, d_1, c_1)}.*
The event sequence E_2 ={0, p_2-t, 2p_2-t, 3p_2-t,...}, therefore E_{S2} = {(∞ ,0, d_2, c_2), (p_2, t, d_2, c_2)}. It is a periodic event sequence with events allowed to jitter. The minimal distance between two events in this event stream is p-t. This happens if one event occurs at the end of the jitter interval and the next event occurs at the beginning of the next jitter interval.
The event sequence E_3 ={0, 0, 0, t, p, p, p, t+p, 2p, 2p, 2p, t+2p,...}. Therefor E_{S3} = {(p, 0, d_3, c_3), (p, 0, d_3, c_3), (p, 0, d_3, c_3), (p, t, d_3, c_3)}. Note that in an event sequence several events can have the same release time.

During the transformation of a event sequence into a event stream the period of the sequence remains.

A real-time analysis algorithm is feasible if all events occur in a lower or equal density than described by the event stream. It is not feasible, if they occur in a higher density than described in the event stream. Using the term event stream, the event function n_e can be calculated easily by:

$$n_i(E_S, I) = \left\lfloor \frac{I - a_i}{p_i} + 1 \right\rfloor \qquad I \ge 0 \qquad (*)$$

Remember that each event stream element describes a periodic event sequence. The maximal number of events in the interval *I* is then simply the sum of all single event stream elements:

$$\forall I \ge 0 \qquad n_e(E_s, I) = \sum_{i=1}^{n} \left\lfloor \frac{I - a_i}{p_i} + 1 \right\rfloor \qquad (*)$$

Example: *For $I < p$ the number of events of $E_{S1}(I)$ is 1, for $p \le I < 2 \cdot p$ the amount is 2 and so on. Consider the situation in the interval $I = t$:*

$$n_e(E_{s2}, p) = \left\lfloor \frac{t - 0}{\infty} + 1 \right\rfloor + \left\lfloor \frac{t - t}{p} + 1 \right\rfloor = 2$$

Each event stream element generates exact one event.

For each task a separate homogenous event stream has to be constructed, because non homogenous event streams are not suitable for real-time analysis. If all the tasks of the same type are independent, the homogenous event streams can simply be merged to one event stream by merging the sets of event stream elements. This event stream represents the worst case density for a single task.

Example: *E_{S1} and E_{S2} are both homogenous event streams. Therefore the merged stream E_{S12} = {(∞ , 0, d_2, c_2), (p_1, 0, d_1, c_1), (p_2, t, d_2, c_2)} = {(∞ , 0, 2000, 900), (4, 0, 2, 2), (4000, t, 2000, 900)}.*

2.1.1 Demand Bound Function

To construct a real-time analysis algorithm it is necessary to calculate the maximum required workload of a processor in a given time interval. This can be provided by the demand bound function, which was first defined by Baruah [3] and Gresser [9] - and later also used by Buttazzo [5].

Definition 15:Demand Bound Function $D_b(I)$ [2]:*The demand bound function $D_b(I)$ denotes the maximum cumulative execution requirement given by events that have both their request time and their deadline within any time interval of length I:*

$$D_b(I) = \sum_{\substack{\forall i \\ I \geq d_i}} n_i(E_S, I - d_i) \cdot c_i \qquad (*)$$

or

$$D_b(I) = \sum_{\substack{\forall i \\ I \geq d_i}} \left\lfloor \frac{I - d_i - a_i}{p_i} + 1 \right\rfloor \cdot c_i \qquad (*)$$

In real-time analysis only events are relevant which have both, their request time *t(e)* and their deadline *t(e)+d_e* within the interval *I*. These are all events occurring in the interval $I - d_e$. The demand bound function matches the event function and is also a monotonic non decreasing function. As a result it is also possible to define an demand stream matching the event stream:

Definition 16: Demand Stream Element D_{bi}: *The demand stream element is a description of a periodic sequence of demands. It consist of a initial finishing time f, the period p and the additional execution demand c.*

$$D_{b_i}(I) = \left\lfloor \frac{I - d_i - a_i}{p_i} + 1 \right\rfloor \cdot c_i \qquad I \geq d_i$$

Definition 17: Demand Stream: *A demand stream is a set of demand stream elements describing the cumulative worst case execution amount of a system.*

To transfer an event stream to its corresponding demand stream, each event stream element is replaced by a demand stream element with an initial finishing time $f_i = a_i + d_i$ and the same costs c_i and deadline d_i of the event stream element. It is not necessary that a demand stream has a corresponding event stream. A demand stream can also model the difference between two alternative event streams with different costs and timing. This concept gives a new formal representation for the demand bound function and is a result of the definitions given earlier in this paper.

2.2. Feasibility Test

A feasibility test or real-time analysis can be constructed by using the demand bound function.

Lemma 2: Processor Demand Criteria: *A system is feasible if the demand bound function is always lower or equal than I [2]:* $\forall I > 0 \qquad D_b(I) \leq I$

The most important problem using the processor demand test is to find the length of the interval *I*. A schedulability test has to consider all relevant events in order to ensure that all tasks will finish before their deadlines. Because $D_b(I)$ is a non continuous function each event defines a relevant test point for the analysis algorithm defined in Section 4.1.. The run-time complexity of the analysis algorithm depends on the length of the interval *I*.

Definition 18: Feasibility Interval: *An time interval $I_{\max}$ is called feasibility interval if*

$$(\exists I > 0) | (D_b(I) > I) \Rightarrow \exists I' : 0 \leq I' \leq I_{max} | (D_b(I') > I')$$

If the processor demand test fails for the time interval *I*, it exists an time interval $I' \leq I_{max}$ for which the processor demand test also fails.

To proof the feasibility of a given task system it is necessary to test only intervals $I < I_{\max}$. Baruah et al. gives such a feasibility interval which was proven in [3]:

Lemma 3: *Let $U = \sum c_i / p_i$ be the maximum used capacity. Than I_{max} is a feasibility interval.*

$$I_{\max} = \frac{U}{1 - U} \cdot \max_{1 \leq i \leq n}(p_i - d_i)$$

If *U* is bounded, the processor demand criteria can be tested in pseudo-polynomial time. In [16] a few better feasibility intervals are given. For all these intervals a problem arises when $D_b(I)$ contains several event streams with a large diversification of periods.

Example: *Consider E_{S1} and E_{S2} of Figure 1. U = 2/4 + 900/4000= 77,5%. The test interval depends only on p_2. It is $I_{max} = (0,775)/(0,225) \cdot (4000 - 2000) = 6889$. So for e_2, 2 test points (at time point 2000 and 6000) are necessary, whereas for e_3 more than 1700 test points are necessary. It is obvious that the complexity of the processor demand test also depends on the different periods of the task set.*

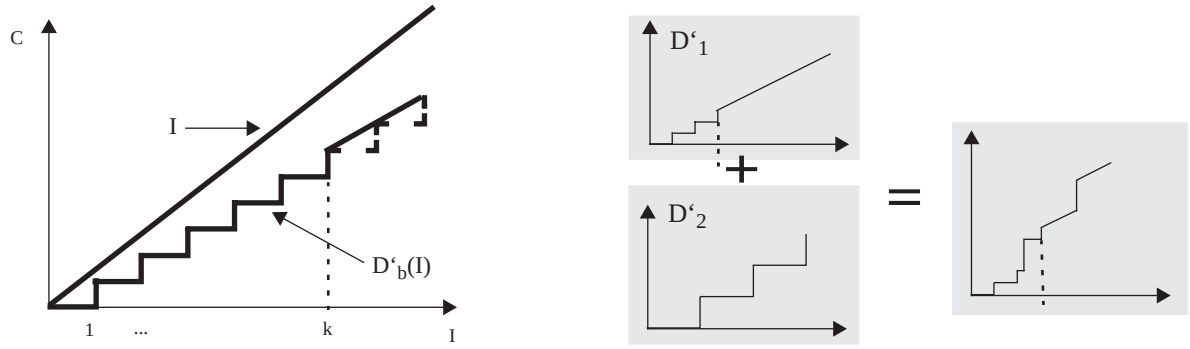

Figure 2: Approximation of the Demand Bound Function by Superposition

3. Approximation by Superposition

To avoid this problem an approximation which is independent from the given periods of the events is needed. The next chapter gives such an approximation.

3.1. Approximated Demand Bound Function

The main idea is to limit the number of test points for each demand stream element separately by constructing a approximated demand stream element function $D'_{b_i}(I)$ and to superpose all approximations to a approximated demand bound function $D'_b(I)$. So for each demand stream element a separate test interval is defined.

Definition 19: Maximum Test interval $I_m(e_i)$: *The maximum test interval $I_m(e_i)$ of the demand stream element e_i is the interval which includes k+1 test points for e_i.*

$$I_m(e_i) = k \cdot p_i + f_i$$

Remember that a demand stream element is described by its initial finishing time f_i and its period p_i.

Definition 20: Approximated Demand Stream Element Function $D'_{b_i}(I)$:

$$D'_{b_i}(I) = \begin{cases} D_{b_i}(I_m(e_i)) + \dfrac{c_i}{p_i} \cdot I - I_m(e_i) & I > I_m(e_i) \\ D_{b_i}(I) & I \le I_m(e_i) \end{cases}$$

As Fig. 2 shows, the approximated demand stream element function is always equal or greater than the demand stream element function. Of course, it is also depending on E_S. Since $c_i/p_i \le 1$ it is only necessary to check all the test points up to $I_m(e_i)$.

Definition 21:Approximated Demand Bound Function $D'_b(I)$: *The approximated demand bound function is a superposition of all approximated demand stream element functions* (Fig. 2)*:*

$$D'_b(I) = \sum_{\forall j} D'_{b_j}(I)$$

Since the amount of each approximated element is at least equal to the amount of the exact element, also the summed amount of the approximated elements is at least equal to the amount of the demand bound function.

The relevant test points of $D'_b(I)$ are all the test points of the elements $D'_{b_i}(I)$. For intervals larger than $I_m(e_i)$ the approximated costs for e_i has to be taken into account at each remaining test interval of the demand stream elements.

Example: *Consider again E_{S1} and E_{S2} of Figure 1: Let $k = 100$. The maximum number of test points $I_m(e_3) = 398$. The next test point is the first test point of e_2, $I_2 = 0 \cdot a_2 + d_2 = 2000$. The remaining test points of e_1 are skipped. The correct value for the demand of I_2 would be $D_b(I_2) = c_2 + 500 \cdot c_1$ and the correct value of $D'_b(I_2)$ is $D'_b(I_2) = c_2 + 500,5 \cdot c_1$.*

The error of the approximation is given by the difference between $D_b(I)$ and $D'_b(I)$. The error of the example is lower than 0.1%. That means feasibility is guaranteed on a processor with 0.1% more capacity than the optimal processor.

Lemma 4::*Let P be a processor with a capacity C(I) and P' a processor with the capacity $C'(I) = C(I) + 1/k \cdot C(I)$. If the feasibility test for P using $D_b(I)$ succeeds (that means $\forall I: C(I) \ge D_b(I)$), the feasibility test for P' using the approximated demand bound function $D'_b(I)$ also succeeds.*
That means:

$$\frac{D'_b(I) - D_b(I)}{D_b(I)} \le \frac{1}{k} = \frac{C'(I) - C(I)}{C(I)}$$

Therefore the error can be bounded by the test limit *k*. A test limit of 100 intervals leads to an maximum error of 1%, a limit of 1000 to an error of 0.1%. The reason is shown in Fig. 3. When the test limit is reached and the approximation begins, the demand already contains $k \cdot c_i$ costs from D_{bi}. The maximum error resulting from the approximation of e_i is limited to $1 \cdot c_i$. In the worst case the ratio between the error and the complete costs is lower than $c_i/k \cdot c_i = 1/k$. The next chapter gives a formal proof for this intuitive conclusions.

3.2. Proof of Correctness

First, it must be proven that the approximated demand bound function (definition 21) is always greater as or equal to the exact demand bound function. This is true, if the approximated demand stream element function as defined in definition 20 is always equal or greater than the real demand stream element function. Figure 2 illustrates the correlation of the two functions. In the following the formal proof of the correlation is given.

Lemma 5:*Let I be an interval with I > 0. Then* $D_{b_i}(I) \le D'_{b_i}(I)$.

For all intervals $I < I_m(e_i)$ the proof is trivial, so it is only necessary to concentrate on the case $I \ge I_m(e_i)$.

Note the definition of the demand stream element (definition 16):

$$D_{b_i}(I) = \left\lfloor \frac{I - f_i}{p_i} + 1 \right\rfloor \cdot c_i \le \left(\frac{I - f_i}{p_i} + 1 \right) \cdot c_i$$

$$= \left(\frac{I_m(e_i) - f_i + I - I_m(e_i)}{p_i} + 1 \right) \cdot c_i$$

$$= \left(\frac{I_m(e_i) - f_i}{p_i} + 1 \right) \cdot c_i + \frac{c_i}{p_i} \cdot (I - I_m(e_i))$$

$$= D_{b_i}(I_m(e_i)) + \frac{c_i}{p_i} \cdot (I - I_m(e_i)) = D'_{b_i}(I)$$

It must also be shown in a further step that this result applies to the demand bound function:

Lemma 6::*If* $D_{b_i}(E_S, I) \le D'_{b_i}(E_S, I)$ *for all intervals I>0, than also* $D_b(I) \le D'_b(I)$.

For each single element, the approximated element function is greater or equal to the real element function. Therefore the sum of all element functions, the approximated demand bound function (definition 21), is always greater or equal to the exact demand bound function (definition 15):

$$\sum D'_{b_j}(I) \ge \sum D_{b_j}(I)$$

$$D'_b(I) \ge D_b(I)$$

It needs to be proven that it is sufficient to check the remaining testing interval if the approximated demand bound function $D'_b(I)$ is used.

Lemma 7::*Let* I, I_{next} *be two consecutive test intervals for* $D'_b(I)$. *If we assume it exists a* ΔI *for which* $I < I + \Delta I < I_{next}$ *and* $D_b(I + \Delta I) > I + \Delta I$ *applies then the approximated demand bound function* $D'_b(I) > I$.

If the test for the exact demand bound function D_b fails at interval ΔI, the test of the approximated demand bound function D'_b fails for the last test interval before ΔI. This last test interval is I. Therefore, if the test fails, it also fails at one of the remaining test intervals.

To proof lemma consider the following inequations

$$D_b(I + \Delta I) > I + \Delta I$$

$$D'_b(I + \Delta I) > I + \Delta I$$

For all demand stream elements i who have reached their test limit, definition 20 leads to the following equation:

$$D'_{b_i}(I + \Delta I) = D'_{b_i}(I) + c_i / p_i \cdot \Delta I$$

For all other elements of the event stream, $I + \Delta I$ has to be between two consecutive test points. This gives $D_{bj}(I + \Delta I) = D_{b_j}(I)$. Therefore it can be followed:

$$D'_b(I) + \sum \frac{c_i}{p_i} \cdot \Delta I > I + \Delta I$$

The utilization $\sum c_i / p_i$ has always to be lower than 100%:

$$D'_b(I) + \Delta I > I + \Delta I$$

$$D'_b(I) > I$$

That is the condition of lemma 7. If the test fails at all, it also fails at one of the remaining test intervals. Accordingly it is only necessary to test these remaining test intervals.

To determine the size of the error an additional lemma is needed:

Lemma 8: *The maximum error between a single approximated demand stream element and the real demand stream element is limited to* c_i: $D'_{b_i}(I) - D_{b_i}(I) \le c_i$

$D_{b_i}(I)$ and $D'_b(I)$ intersect at each test interval of $D'_{b_i}(I)$, therefore both functions have the same value at the end of the test interval. The cost difference between two consecutive test intervals of $D_{b_i}(I)$ is c_i. Therefore the maximum difference between both functions is also c_i.

Note first:

$$D_{b_i}(E_S, I) = \left\lfloor \frac{I - f_i}{p_i} + 1 \right\rfloor \cdot c_i$$

$$= D_{b_i}(I_m(e_i)) + \left\lfloor \frac{I - I_m(e_i)}{p_i} \right\rfloor \cdot c_i$$

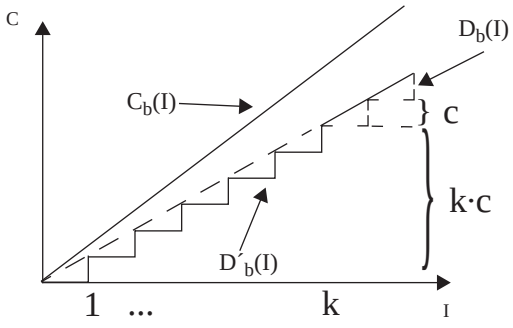

Figure 3: The Error of the Approximation

and then remember the definition of $I_m(e_i)$. Both functions intersects at this interval. With this knowledge it is easy to proof:

$$\begin{aligned} D'_{b_i}(I) - D_{b_i}(I) &= D_{b_i}(I_m(e_i)) + \frac{c_i}{p_i} \cdot (I - I_m(e_i)) - D_b(I) \\ &= \frac{c_i}{p_i} \cdot (I - I_m(e_i)) - \left\lfloor \frac{I - I_m(e_i)}{p_i} \right\rfloor \cdot c_i \\ &\le \frac{c_i}{p_i} \cdot (I - I_m(e_i)) - \left\lceil \frac{I - I_m(e_i)}{p_i} - 1 \right\rceil \cdot c_i \\ &\le \frac{c_i}{p_i} \cdot (I - I_m(e_i)) - \left(\frac{I - I_m(e_i)}{p_i} - 1 \right) \cdot c_i \\ &= c_i \end{aligned}$$

Therefore the maximum error of $D'_{b_i}(I)$ is limited to c_i.

Using this result the error of the approximated demand bound function $D'_b(I)$ can be bounded. One upper bound is the sum of the errors of the single element functions $D'_{b_i}(I)$. If the interval is shorter than the maximum test interval $I_m(e_i)$, the error for this element is *0*. The maximum error of the demand bound function is bounded by:

$$D'_b(I) - D_b(I) \le \sum_{e_i \mid I > I_m(e_i)} c_i$$

The minimum cost of the demand bound function is limited by:.

$$\begin{aligned} D_b(I) &\ge \sum_{e_i \mid I > I_m(e_i)} D_{b_i}(I_m(e_i)) \\ &= \sum_{e_i \mid I > I_m(e_i)} \left\lfloor \frac{k \cdot p_i + f_i - f_i}{p_i} + 1 \right\rfloor \cdot c_i \\ &= \sum_{e_i \mid I > I_m(e_i)} (k+1) \cdot c_i \ge \sum_{e_i \mid I > I_m(e_i)} k \cdot c_i \end{aligned}$$

With these two bounds, the proof of lemma 4 is easy:

$$\frac{D'_b(I) - D_b(I)}{D_b(I)} \le \frac{\sum_{I > I_m(e_i)} c_i}{\sum_{I > I_m(e_i)} k \cdot c_i} = \frac{1}{k}$$

Therefore, the overall approximation error is bounded by *1/k.*

4. Approximated Feasibility Test

4.1. Test Algorithm

For an efficient testing algorithm, it is not necessary to calculate $D_b(I)$ for every test interval separately. Fig. 4 shows the complete approximation feasibility test algorithm.

First it initializes the test list with the first instance of all demand stream elements, using their finishing times f_i as initial time point. Then it processes this list in ascending order of the test intervals. For each event it adds the corresponding costs c_i to the cumulated costs. It tests if the cumulated costs are higher than the computing time for the actual test interval, then the test failed. If the maximum number of test intervals of this element is not reached, the next instance of this element is added to the test list. It has a distance p_i to the actual test interval. Therefore at each time, the test list contains at maximum one test point of each demand stream element. If the maximum number of test points for an element is reached, its utilization c_i/p_i is added to U_{ready}. In this case the next test point is not added to the test list. The approximated costs $approx_i$ are also added for each test interval. The calculation of these costs is given by

$$approx_i = (I_{act} - I_{old}) \cdot U_{ready}$$

ALGORITHM ApproxFeasabilityTest
INPUT: ListOfEvents {e_i}, testLimit
$U = \sum_i c_i/p_i$
IF U>100% => not feasible
$I_{max} = U/(1-U) \cdot max_{1 \le i \le n}(p_i - d_i)$
// Test interval from [3]
$\forall e_i \in D_b$: testlist.add(f_i, e_i);
WHILE ($I_{act} \le I_{max} \vee Liste \ne \{\}$)
  i = testlist.getNextDemand();
  I_{act} = testlist.intervallForDemand(i)
  $D'_b = D'_b + c_i + (I_{act} - I_{old}) \cdot U_{ready}$
  IF ($D'_b > C_b(I)$) => not feasible
  IF ($I_{act} < (k-1) \cdot p_i + f_i$)
    testlist.add($I_{act} + p_i, e_i$)
  ELSE $U_{ready} = U_{ready} + c_i/p_i$
  $I_{old} = I_{act}$
END WHILE
=> feasible

Figure 4: Superposition Test Algorithm

4.2. Complexity of the Algorithm

The complexity of the original problem is unknown. Only pseudo-polynomial solutions are know so far[3].

Let *n* be the number of demand bound elements and *k* the maximum number of testing points for each element.

ALGORITHM Chakraborty
INPUT: ListofTasks {e_i}, testLimit
$I_{max} = U/(1-U) \cdot max_{1 \le i \le n}(p_i - d_i)$
// Test interval from [3]
$K = \frac{I_{max}}{testLimit}$
FOR ($t \leftarrow 1$ to $\lfloor I_{max}/K \rfloor + 1$)
  IF ($D_b(T, f_{max} + t \cdot K) > f_{max} + (t-1) \cdot K$) THEN
    => not feasible
END FOR
=> feasible

Figure 5: Algorithm Chakraborty

For the sporadic task system the number of demand bound elements is equal to the number of tasks. The value k is a selectable variable which affects both, the complexity and the error of the algorithm. Therefore a trade-off between the run-time of the algorithm and the error is possible. The error ε is $\varepsilon = 1/k$. Each test point has to be inserted in a sorted list ($O(\log n)$). So the complexity of the approximated feasibility test is $O(n \cdot \log n \cdot 1/\varepsilon)$. The given analysis error ε gives a requirement to the processor used in the final implementation of the system. To guarantee all deadlines of the considered system this processor is at most ε percent faster than the unknown optimal processor.

The algorithm of Baruah et al. [3] has pseudo-polynomial complexity depending on the number of tasks, the overall utilization and the granularity of the periods. The complexity of the algorithm given by Chakraborty et al. [6] has the same complexity as the algorithm presented in this paper, but uses, as outlined in the introduction, a different kind of error.

5. Results

The test algorithm is compared with the precious approach of Chakraborty et al. [6]. Fig. 5 contains a simplified version of Chakraborty's algorithm. Two experiments are described in this section to validate the theoretical results and to give an impression how the new approximation algorithm works. To make the two algorithms comparable Chakraborty's algorithm was re-implemented in the same framework as the new superposition approach. Therefore, both algorithms are implemented in Java running on a 1 GHz PowerPC on MacOS X.

Two case studies are considered: The first example comes from [4] and models the *Olympus Attidude and Orbital Control System* for Satellites. This example contains 10 periodic and 4 sporadic tasks and is given in Tab. 1.

Task	Period/Min. Distance	Start time	WCET	deadline
τ_1	50	0	0.28	9.0
τ_2	10	0	1.76	10.0
τ_3	200	50	2.13	14.0
τ_4	200	150	1.43	17.0
τ_5	200	0	1.43	17.0
τ_6	100	0	1.43	24.0
τ_7	100	50	8.21	50.0
τ_8	200	50	52.84	200.0
τ_9	1000	200	5.16	400.0
τ_{10}	1000	200	6.91	900.0
τ_{11}	0.96	0	0.18	0.63
τ_{12}	62.5	0	3.19	30.0
τ_{13}	100.0	0	4.08	100.0
τ_{14}	187.0	0	2.5	187.0

Table 1: Task System of the *Olympus Attidude and Orbital Control System*

Error	Time (in ms 100 runs)		Example
	New	Chakraborty	
50%	190	failed	SAT
5%	730	failed	SAT
1%	1713	failed	SAT
0.5%	2744	failed	SAT
0.05%	11423	22755	SAT
0.02%	22273	58081	SAT
0.01%	22244	116825	SAT
10%	failed	failed	PALM
5%	629	failed	PALM
0.5%	3961	failed	PALM
0.2%	5046	7885	PALM
0.1%	4588	16035	PALM
0.05%	4999	30039	PALM
0.02%	5207	82156	PALM
0.01%	4974	181786	PALM

Table 2: Experimental Results

The second example was originally given by Ma and Shin [14] and can also be found in [11]. The model describes a *Palmpilot application* containing 13 different tasks. All these tasks have deadlines equal to their periods. To define a harder problem for the experiment we have set the deadline for task τ_7 to 100 ms instead of 150 ms in the originally model.

Let us consider the experiment using the task set of the *Olympus Satellite System*. It is shown by the first seven lines in *Tab. 2* indicated by the phrase *SAT*: If we assume a approximation error of 0.05% the new approximation finishes the schedulability test after 114 ms (100 runs in 11423 ms) while the algorithm of Chakraborty needs 228 ms. Despite the fact that, for the task set a feasible schedule exists, Chakraborty's algorithm fails in cases where he uses a chosen error of 50%, 5%, 1% and 0.5%. In all these cases the new superposition algorithm achieves results. Therefore it is possible by the superposition algorithm to perform a schedulability analysis in 17 ms with an expected error of 1%.

The effort for the new algorithm - apart from the overhead for the approximation- is always lower than or equal to the effort for the exact algorithm, even with a very low error. The effort for Chakraborty's algorithm grows with sinking error. Therefore, the density of test intervals can be higher than necessary for the exact algorithm.

For the palmpilot example, the results goes in a equal direction. The new algorithm accepts the task set with an error of 5%, whereas the previous approach needs a stricter error (the experiments have shown that an error of 0.2% is sufficient). If both algorithms run with an given error of 0.01%, which is near the optimal processor, the superposition approximation needs 49 ms compared to 1817 ms needed by Chakraborty's algorithm.

Figure 6 visualizes the reason for this behavior. It contains a part of a test run of both algorithm, compared with

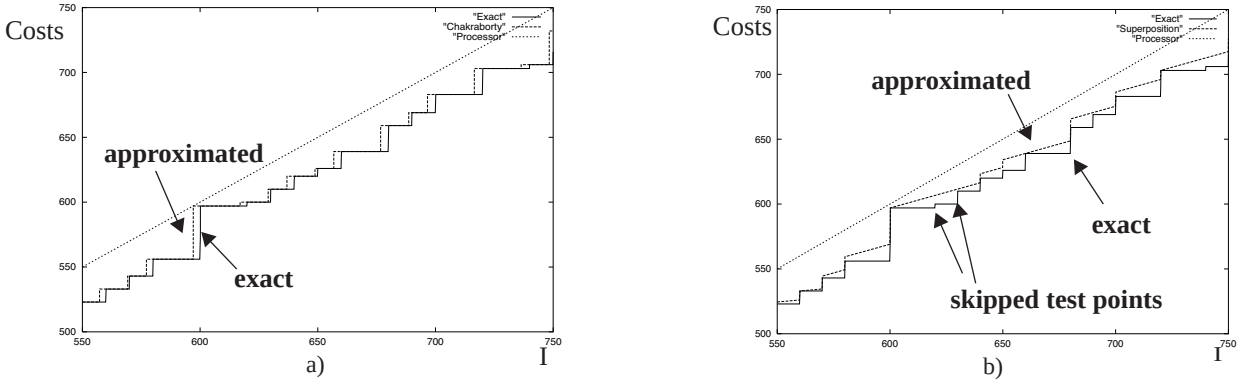

Figure 6: a) Palmpilot Experiment comparing the algorithm by a) Chakraborty with the b) Superposition approach

the exact solution. In a) a run of Chakraborty's algorithm with an error of 0.1% is shown. Also the error is to small to skip any test points in this part of the run, the approximation needs the costs always earlier than the exact solution. Therefore it needs a processor with more capacity. In b) the same part of the run of the superposition algorithm using an error of 5% is shown. The error allows skipping some test points. Despite this, on the critical test points the approximation is closer to the exact solution than the approximation of Chakraborty.

6. Conclusion

In this paper we have presented and proved a fully polynomial time approximation scheme (FPTAS) for the sporadic task system. This analysis of sporadic tasks has an complexity $O(n \cdot \log n \cdot k)$ where k is an arbitrary amount of test intervals. The error k is bounded in terms of the capacity of the optimal processor and a trade-off between the error and the analysis time is possible.

In contrast to the previous work it is possible to analyze systems which have a large diversification of the periods.

7. References

[1] S. K. Baruah. *Dynamic- and Static-priority Scheduling of Recurring Real-Time Tasks,* Journal of Real-Time Systems, 24, 93-128, 2003.

[2] S. Baruah, D. Chen, S. Gorinsky, A. Mok. *Generalized Multiframe Tasks.* The International Journal of Time-Critical Computing Systems, 17, 5-22, 1999.

[3] S. Baruah, A. Mok, L. Rosier. *Preemptive Scheduling Hard-Real-Time Sporadic Tasks on One Processor*. Proceedings of the Real-Time Systems Symposium, 182-190, 1990.

[4] A. Burns, A. Wellings. *HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems*. Elsevier, Oxford, 1995.

[5] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[6] S. Chakraborty, S. Künzli, L. Thiele. *Approximate Schedulability Analysis*. 23rd IEEE Real-Time Systems Symposium (RTSS), IEEE Press, 159-168, 2002.

[7] S. Chakraborty, T. Erlebach, L. Thiele. *On the Complexity of Scheduling Conditional Real-Time Code*. TIK Report Nr. 107, ETH Zürich, 2001.

[8] D. Chen, A. Mok, S. Baruah. *On Modeling Real-Time Task Systems*. Lectures on Embedded Systems: Proceedings of The European Educational Forum School on Embedded Systems, in Lecture Notes in Computer Science, No. 1494, 153-169, Springer Verlag, 1998.

[9] K. Gresser. *Echtzeitnachweis Ereignisgesteuerter Realzeitsysteme*. Dissertation, VDI Verlag, Düsseldorf, 10(286), 1993. (in german).

[10] J. Hromkovic. *Algorithmics for Hard Problems*. Texts in Theoretical Computer Science, Springer Verlag, 2003.

[11] T.M. Lee. J. Henkel. W. Wolf. *Dynamic Runtime Re-Scheduling Allowing Multiple Implementations of a Task for platform-based Designs*. 296-301. Proceedings of the Design and Test Conference in Europe, DATE, IEEE Press, 2002.

[12] C. Liu, J. Layland. *Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments*, Journal of the ACM, 20(1), 46-61, 1973.

[13] J. Leung, M. Merril. *A Note on Preemptive Scheduling of Periodic Real-Time Tasks*, Information Processing Letter, Volume 11, p. 115-118, 1980.

[14] C. Ma, K. Shin. *A user-customizable energy-adaptive combined static/dynamic scheduler for mobile communications*. Proceedings of the Real-Time Symposium, 2000

[15] M. Spuri. *Analysis of Deadline Scheduled Real-Time Systems*, Rapport de Recherche RR-2772, INRIA, Le Chesnay Cedex, France 1996

[16] J.A. Stankovic, M. Spuri, K. Ramamritham, G.C. Buttazzo. *Deadline Scheduling for Real-Time Systems EDF and Related Algorithms*. Kluwer Academic Publishers, 1998