Approximative Real-Time Analysis

DISSERTATION

zur Erlangung des akademischen Grades eines
Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften und Informatik
der Universität Ulm

vorgelegt von
Karsten Albers
aus Erlangen

Institut für Eingebettete Systeme / Echtzeitsystem, 2011

ulm university universität ulm

Amtierender Dekan:	Prof. Dr.-Ing. Klaus Dietmayer Universität Ulm, Deutschland
Gutachter:	Prof. Dr.-Ing. Frank Slomka Universität Ulm, Deutschland
Gutachter:	Prof. Dr. Helmuth A. Partsch Universität Ulm, Deutschland
Externer Gutachter:	Prof. Dr. Lothar Thiele ETH Zürich, Schweiz
Tag der Promotion:	08.04.2011

Contents

List of Figures

List of symbols

descriptor	meaning	First defined in
τ	task	chapter2
Γ	task set	chapter 2
$\tau_{i,j}$	j- th job of a task τ_i	chapter 2
p	period	chapter 2
c	execution-time	chapter 2
c^+	worst-case execution-time	chapter 2
c^-	best-case execution-time	chapter 2
d	relative deadline	chapter 2
U	utilization	chapter 2
Δt	interval	section 2.1
r	response time	section 2.1
$hp(\tau)$	set of tasks with a higher priority than τ	section 2.1
j	jitter	section 2.1
S	scheduling point set	section 2.1
$\delta()$	demand bound function	section 2.2.2
$\chi()$	capacity bound function	section 2.2.2
Δt_{max}	maximum test interval	section 2.2.3
Δt_{LCM}	least-common multiple of periods interval	section 2.2.3
$\mathscr{B}()$	busy period function	section 2.2.3
n	number of tasks	section 2.2.4
$\eta()$	event bound function	section 2.3.2
Θ	periodic event sequence	section 2.3.2
θ	event element	section 2.3.2
a	offset / initial interval	section 2.3.2
$\psi()$	interval bound function	section 2.3.2
s	minimum separation distance	section 2.3.4
α	arrival curve	section 2.3.6
α^u	upper arrival curve	section 2.3.6
α^l	lower arrival curve	section 2.3.6
β	service curve	section 2.3.6
β^u	upper service curve	section 2.3.6
β^l	lower service curve	section 2.3.6

descriptor	meaning	First defined in
γ	either arrival or service curve	section 2.3.6
ρ	resource	section 2.3.6
$\inf()$	infimum	section 2.3.6
$\sup()$	supremum	section 2.3.6
$\delta'()$	approximated demand bound function	section 3.1
ε	approximation error	section 3.1
k	number of exact evaluated test intervals	section 3.1
C	capacity	section 3.2
$\rho()$	request bound function	section 5.1
$\mathscr{E}()$	exceeding costs function	section 5.2
$\hat{\Theta}$	event spectrum	section 7.2
$\hat{\Theta}^k$	approximated event spectrum with k exact evaluated test intervals	section 8
$\hat{\Theta}^+$	upper event spectrum	section 7.2
$\hat{\Theta}^-$	lower event spectrum	section 7.2
$\hat{\theta}$	event spectrum element	section 7.2
n	limitation (amount of costs / number of events)	section 7.2
L	limitation (length of interval)	section 7.2
f	slope	section 7.2

CHAPTER 1

Introduction

An average car of today has a large number of embedded systems handling applications. The requirements to reduce fuel consumption combined with the pollution reduction leads to complex motor management systems. A growing number of driver assistance systems like the break management, the electronic stabilization program, the potential collision detection and so on, are integrated in modern cars.

The result of all these new features is the integration of up to 70 ECUs (electronic control units) with hundreds of functions in a modern car [120] ([121] speaks of up to 100 ECUs) and that the embedded systems are responsible for often more than 20% of the total costs of a car. All these embedded car systems are connected and communicate with each other. In future there will be a lot more of such systems like car-to-car communication to get a complete picture of the traffic ahead.

Cost reduction is a very important topic for today's automotive industry. One way to do this would be to reduce the number of different systems and to substitute expensive systems with inexpensive ones, if possible. Nowadays, many of the systems build in cars are designed independently of each other, often from different suppliers. They form a separate unit of hardware and software. In the future, the integration of several applications on one system will be required. For example AUTOSAR is an approach of the automotive industry in this direction [120]. It allows the separation of the functionality from the ECUs and enables therefore the integration of functionality from different vendors on the same ECUs.

But, as many of these applications can be critical for safety, we need to be able to rely on these kinds of systems. It is not only necessary that these systems always deliver the correct results but it is required that they do this within the available time. For such real-time systems methods are required to prove and predict reliability. The best would be to have formal methods that can proof mathematically the real-time requirements. This is not an easy task, especially if several embedded systems are connected and work together or interfere with each other.

The module-based design processes make it possible to handle the complexity in software and hardware designs. Systems are constructed using a set of closed modules. These modules can be designed and developed separately. The purpose of modularization is to split the challenging job of designing the whole system into multiple smaller jobs. Another purpose is to allow the reuse of modules in different designs. Also for using IP (Intellectual property) components, which are developed by third-party vendors, it is necessary to have a module-based design concept.

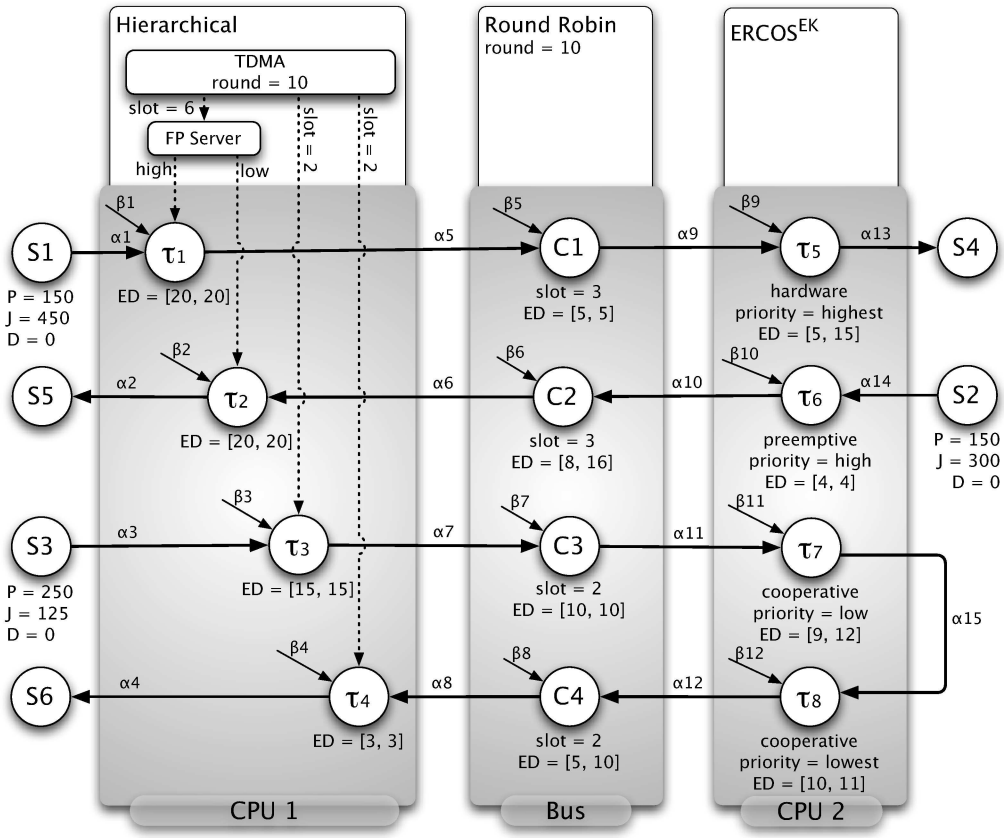

FIGURE 1.0.1. Example of a distributed hard-real time system published in [77]

For each set of modules a well-defined interface-concept for connecting the different modules is required. For developing real-time systems with such a modular approach it would be interesting to have a concept for analyzing the system which can handle the real-time aspects of the different modules separately. With this concept it is possible to hide the details of the scheduling and its real-time analysis inside the modules and to abstract the interface from the construction of the modules. For the global real-time analysis it is necessary to propagate the analyses results of the different modules in an abstract way through the system. The global analysis can than be build by connecting the local analyses of the single modules by well-defined interfaces. Therefore it is essential to have an expressive and efficient interface describing the influence in time of one module to the next module. One aspect of this interface is the timing description of events, which are produced by one module to activate the next following module. Another aspect is the remaining computation capacity for the next module left over by the previous modules.

Consider for example a system as shown in figure 1.0.1. Chains of tasks τ_i that can be located on different resources ρ_i process the activations. There are three chains of tasks. The tasks are distributed on two processors and one connecting bus. Each of these tasks can be considered as one separate module. The resources ρ can be processors, dedicated

hardware or the communication network. The tasks are activated either by events from other tasks or events from dedicated sources (in the model). The resource CPU2 uses a fixed-priority scheduling. The available capacity for each task on CPU2 depends on the totally available capacity for CPU2 and on the consumed capacity of all tasks with a higher priority on the same resource. See chapter 9 for a detailed description of the example.

First, we need an accurate but compact model to describe the events in the network. We need an event model to describe the incoming and outgoing event streams and we need a model for the existing and remaining capacities. Second, we need a methodology to analyze each task together with his incoming streams. This methodology requires not only to proof the real-time constraints of the task itself but also to calculate the outgoing streams.

Having many different modules and the requirement of cost reduction in mind we need an optimization process to find the cheapest set of hardware components that can handle all the tasks and the best distribution of the tasks between hardware and software components that keeps the costs low but still achieves all necessary real-time requirements. For the optimization it is necessary to analyze thousands of different possible distributions and therefore to perform a complex real-time analysis for each of them. The previously available real-time analysis approaches are either only sufficient or have a exponential or pseudo-polynomial complexity. Sufficient means that there are many schedulable systems, especially those leading to a high utilization of the systems, which cannot be classified correctly with these approaches. Therefore an efficient method for real-time analysis is required that reduces the run-time of the previous methods.

The purpose of this thesis is to develop such a new efficient way for real-time analysis. The idea is to allow a faster analysis by using a bit of uncertainty. An approximation algorithm solves this. The advantage of this approximation is a lower (polynomial) complexity than the one of previous results and that it can guarantee both the maximum run-time and the degree of uncertainty. In chapter 3 we propose such an approximation for the optimal EDF scheduling, in chapter 4 we propose fast exact analysis algorithms based an the approximation algorithms using a dynamically changing degree of exactness and in chapter 5 we extend both kinds of algorithms to static priority scheduling.

Additionally, we contribute to the theory of event models. The real-time calculus and the sub-additive and super-additive event bound functions (and also service / capacity bound functions) on which the real-time calculus is based are the key concept for the schedulability analysis. The answer on the question how many events can occur at most / at least within any possible interval of length Δt for each interval length Δt leads to a integrated theory on schedulability analysis. These functions extract the worst-case situations of all possible concrete schedules into one single description. We call this concept event spectrum, because an event spectrum contains all possible worst-case event densities like the light spectrum contains all possible wavelengths of the different colors of light. Having an efficient and compact description for event spectra which allows a fast calculation of the values for each interval leads to an efficient real-time analysis. Many proposed models in the real-time community are concrete descriptions of event spectra or can be interpreted

as a description of event spectra. Examples are the event stream model [61] (introduced in section 2.3.2), the periodic model with jitter and minimum separation distance (introduced in section 2.3.4), the concrete description and the approximation of the real-time calculus curves (introduced in section 2.3.6). Also most of the analysis proposed for the periodic model, the periodic model with jitter and the recurring real-time task model are fulfilling the conditions of event spectra. Of course our approximation and all the analysis algorithm proposed for them in chapter 3, chapter 4 and chapter 5 are based also on event spectra.

In this work we propose a new concrete description for event spectra, the hierarchical event spectra model. It is a general model that overcomes some limitations of the event stream model and is especially suitable for modeling all kind of bursts efficiently.

We have already published several ideas and concepts proposed in this thesis. In [5] we have presented the superposition approximation analysis for EDF systems. An early idea of the approximation was developed in our master thesis [1]. In [6] we have extended this analysis to new fast exact analysis and proposed the dynamic error test and the all-approximation analysis. Also we have outlined there the relationships of the superposition analysis to previous work. This covers a sufficient analysis approach proposed by U. Devi [46] and a proposed approximation for the real-time calculus approach [37, 38]. We have proposed our extension of the approximation to static priority scheduling and also of a fast exact analysis algorithm to static priority scheduling [3]. A first version of the hierarchical event spectra model, which we will introduce in chapter 7, has been published in [2], an advanced version in [4].

CHAPTER 2

Related Work

Despite that the first fundamental work in the area of schedulability analysis is more than 30 years old, many questions in this area are still open. Most achievements have been made in the past 10 to 15 years. Nearly all work in the area cites the seminal work of Liu and Layland [88]. Liu and Layland consider a simple task model that consists of a set of independent preemptive tasks bounded on one processing element. Each task is described by a worst-case execution time and a deadline. A task is invoked multiple times and the invocation rate is described by a single period only. The work is restricted to task sets in which the deadline of each task is equal to the period of its invocation rate. So, a new invocation can only occur in this model after the processing of the previous invocation has finished.

Each invocation is called an event of the task and leads to one execution of the task called a job of the task. The event pattern of a task describes how events of this task can occur. It is assumed in the model that in the worst-case, events of all tasks can occur concurrently. Liu and Layland proved for this model the optimality of the deadline monotonic priority assignment for static priority scheduling (tasks with a smaller deadline gets a higher priority, those with a longer deadline a lower priority). They also proved the optimality of earliest deadline first (EDF) scheduling; a scheduling with dynamic priorities, in the sense that every task set that cannot be scheduled by EDF holding all of its deadlines cannot be scheduled by any other scheduling strategy. Additionally, they give a sufficient schedulability condition for the deadline monotonic assignment and proved that systems using preemptive EDF scheduling meet all their deadlines unless they are overloaded, e.g. their utilization exceeds 100%.

Unfortunately, most of these results are no longer valid when the restrictions of the task model are relaxed. The utilization conditions are still valid in case of extended deadlines but they are no longer valid for tasks having deadlines smaller than their corresponding periods. For such task sets the optimality condition of EDF remains true but the RM priority assignment has to be exchanged with the deadline monotonic assignment (DM) [84], giving those tasks a higher priority having a smaller deadline assigned to the task. Note that the RM priority assignment is only a special case of the DM priority assignment in case of the simple task model having only tasks with a deadline equal to the task period.

The schedulability analysis tells whether a given task set with a given scheduling keeps always all of its deadlines. The result of the schedulability analysis is independent of the concrete schedule and the concrete stimulation of a system as long as both are within the

given bounds of the task set and the scheduling scheme. Overviews of this area are given in [33, 34, 82, 90].

We have to distinguish between the analysis of distributed and of uni-processor systems. Distributed systems can be distinguished into multi-processor systems and heterogeneous distributed systems. In multi-processor systems there exist no fixed distribution of the tasks on the single processors and also no or only few dependencies between the tasks. This leads to a centralized scheduling scheme. In [7, 11, 15, 36, 54] approaches for different scheduling schemes with or without task migration are introduced. We will not consider multi-processor systems without a fixed task distribution despite that some of the proposed approaches are useful also for this area [11, 54].

For the analysis for heterogeneous distributed systems the consideration of dependencies between the tasks is required. Normally system architecture with a fixed distribution of the tasks on the processing units of the system is used. The distribution (and partitioning) of the system can be done separately for the analysis by the developer or with an optimization process [29, 44, 45, 48, 49]. The schedulability analysis can then use a fixed distribution of the tasks. The holistic scheduling analysis [130], an extension of the response-time analysis for single processor systems, which will be explained in the following section, uses the jitter to cover the dependencies between the tasks. It was improved and extended by several subsequent approaches [64, 97, 99, 100, 112, 114, 115]. To cover dependencies between tasks on the same processing unit the transaction model [129] was developed. The key concept is to group dependent tasks into transactions. The approached was improved and combined with the holistic scheduling analysis in [20, 98, 99, 111, 112, 113]. Approaches for calculating certain kinds of task dependencies are for example given in [65, 74, 75].

In several approaches the analysis of task sets with special characteristics have been considered. This includes for example task sets with resource constraints [85], mixed time- and event-activated tasks [107, 108, 109], self-suspending tasks [116] and task sets with selectable offsets [59]. All these approaches are specialized to cover one characteristic of the system only.

General holistic approaches for the schedulability analysis for distributed systems are given for example with the extended periodic model [50, 62, 65, 69, 70, 71, 72, 110, 117, 118] and the real-time calculus approach [31, 39, 76, 78, 95, 127, 128, 131, 132, 133]. These approaches, and there relationship to this work, are introduced later in detail.

In the following sections we will give a closer look on the schedulability analysis for uni-processor systems. The schedulability analysis covers the worst-case situations, especially the worst-case densities of events. Relaxing these worst-case situations will preserve the schedulability condition. These circumstances allow the integration of tasks which do not have a constant invocation rate or for which the event pattern is not known. Those tasks are called sporadic tasks. For these tasks the minimum time between two events of the task is used as period in the schedulability tests. Note, that this might lead to a very pessimistic analysis for tasks having a minimum distance between events that is much lower than the average distance between consecutive events of the task.

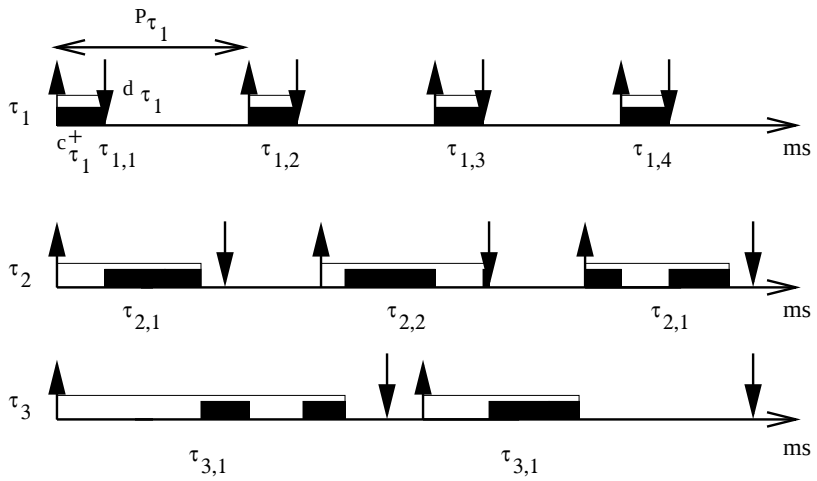

FIGURE 2.0.1. Example task set

Let Γ be a task set having tasks $\tau \in \Gamma$ with $\tau = (p_\tau, c_\tau^+, d_\tau)$ and $\tau_{i,n}$ be the n-th job of task τ_i. Let d_τ be the relative deadline, c_τ^+ be the worst-case execution time and p_τ be the period of (or minimum distance between) the events activating τ. Let $U_\tau = \frac{c_\tau^+}{p_\tau}$ be the utilization of task τ and $U_\Gamma = \sum_{\tau\in\Gamma} U_\tau = \sum_{\tau\in\Gamma} \frac{c_\tau^+}{p_\tau}$ be the utilization of the complete task set. A task set is called rate monotonic if for all tasks τ in the task set Γ the period is equal to the deadline of the task $(\forall \tau \in \Gamma : p_\tau = d_\tau)$ and the task with the smaller period/ deadline gets the higher priority.

2.1. Schedulability analysis for task sets with static priorities

Let us first consider the previous results in the area of fixed priority scheduling.

THEOREM 2.1.1. *[88] Let $n = |\Gamma|$ be the number of tasks τ in Γ. All tasks of an rate monotonic task set meet their deadlines if*

$$U_\Gamma \leq n(\sqrt[n]{2} - 1)$$

PROOF. See [88]. □

The analysis guarantees the schedulability of task sets having utilization not larger than the given bound. For large numbers of tasks in the task set e.g. $n \to \infty$ the test bound will converge to $\ln 2$, that is approximately a utilization of 69.3%. For task sets with utilizations larger than 69.3% it is uncertain whether they meet all deadlines or not. Therefore the analysis is only sufficient.

This first schedulability condition for RM scheduling of Liu and Layland was improved several times. In [42] the test bound was improved by considering possible relationships between the periods of different tasks of the task sets. The results of [102, 63, 89] are based on a similar idea. The latest result in this area is the Hyperbolic Bound (HB), a schedulability condition proposed by Bini et al. [22, 23, 25, 27, 28]:

THEOREM 2.1.2. *A rate monotonic (RM) task set (deadline equal to the periods of the tasks) is schedulable with fixed priority scheduling and a RM priority assignment if*

$$\prod_{\forall \tau \in \Gamma} (U_\tau + 1) \leq 2$$

PROOF. See [27] □

This bound is better than the previous bounds and improves the acceptance ratio up to $\sqrt{2}$ compared to the bound proposed by Liu and Layland. All these schedulability bounds can be evaluated fast even for large task sets. Unfortunately, they are only sufficient, e.g. not all schedulable task sets can be recognized as such by the test. Additionally, the bounds mentioned are only valid for the restricted RM task model.

For an analysis which is both sufficient and necessary the worst-case response-time analysis was developed initially by Joseph and Pandya [73] and later improved in [9]. It calculates the worst-case response times of each task separately by using a fixed-point iteration, taking the interferences by higher priority tasks into account:

THEOREM 2.1.3. *[73] For the worst-case response time r_τ of a task $\tau \in \Gamma$ is given by the smallest value for r_τ fulfilling the following equation:*

$$r_\tau = c_\tau^+ + \sum_{\forall \tau' \in hp(\tau,\Gamma)} \left\lceil \frac{r_\tau}{p_{\tau'}} \right\rceil c_{\tau'}^+$$

PROOF. The fixed-point iteration is necessary because the response time depends on the number of tasks interfering within this time, which again depends on the response-time itself. The calculation would start with e.g. the execution time of task τ as initial value for the response time. All interferences within this execution time are then considered and the initial response time is extended by the delay caused by these interferences. For a fixed-priority scheduling scheme only interferences by tasks with a higher priority are relevant ($hp(\tau,\Gamma)$). In the worst case each of these interferences causes a delay with the worst-case execution time of the task causing the interference, as the worst-case length of the delay. Extending the calculated initial response time by these delays leads to a new (longer) response time, that can result in new additional interferences. These interferences cause additional delays leading to a longer response time and so on. The fixed-point iteration finishes when there are no new interferences or the response time of a task exceeds its deadline. □

The response time analysis was the origin for a lot of work extending this result. The introduction of a jitter was necessary to model variable stimuli. With jitter the arrival of events is still assumed to be periodic in general but single events can arrive a bit earlier or later compared to the strict periodic arrival scheme. The interval in which these earlier or later arrival can happen is denoted jitter interval. Therefore the stimuli of a task τ is described by the period p and the jitter j. The events generally occur periodically but the arrival of each event can vary within j. Therefore the average distance between two events is p. Due to the jitter the minimum distance between two events can be $p - j$ and the maximum distance can be $p + j$.

A jittering high-priority task can cause more interference to a lower-priority task than a non-jittering periodic high-priority task. Therefore the introduction of the jitter leads to a modified worst-case response-time calculation. The number of interferences $I(\Delta t, \tau)$ of a task τ within an interval Δt can be calculated by:

$$I(\Delta t, \tau) = \left\lceil 1 + \frac{\Delta t - (p - j)}{p} \right\rceil = \left\lceil 1 + \frac{\Delta t + j - p}{p} \right\rceil = \left\lceil 1 + \frac{\Delta t + j}{p} - \frac{p}{p} \right\rceil = \left\lceil \frac{\Delta t + j}{p} \right\rceil$$

Therefore the introduction of jitter leads to the following modified response time analysis:

THEOREM 2.1.4. *[9] For a task τ with a period p_τ a jitter j_τ and a worst-case execution time c_τ^+ the response time is given by the minimum value for r_τ fulfilling:*

$$r_\tau = c_\tau^+ + j_\tau + \sum_{\forall \tau' \in hp(\tau,\Gamma)} \left\lceil \frac{r_\tau + j_{\tau'}}{p_{\tau'}} \right\rceil c_{\tau'}^+$$

PROOF. Follows directly out of the previous considerations. □

These results are only valid for task sets in which all tasks have a deadline smaller or equal than their period. In case of larger deadlines, the worst-case response time is not necessarily the response time of the first job of a task. A job having a response time being longer than the period of the task can finish its execution later than the arrival of the next following job. It can postpone the execution of this following job and therefore can cause an even longer response time for this job. The previous analysis can be extended to solve also these arbitrary deadline systems [79, 130]. The idea is to calculate the response time of the first, second, third, ... job of a task until a job is found that finishes before the invocation of the next following job. The resulting response time of the task is than the maximum of all these job response times.

THEOREM 2.1.5. *[79, 129] Let $\tau_{i,q}$ be the first job of task τ_i having a response time smaller than the invocation time of the next following event of τ_i ($r_i \leq (q+1)p_i$) in a set of jobs following a simultaneous release of a job of all tasks in the task set. The response time r_i of τ_i is given by:*

$$r_i = max_{n \leq q}(j_\tau + r_n - (n-1)p_\tau)$$

$$r_n = n \cdot c_\tau^+ + \sum_{\forall \tau' \in hp(\tau,\Gamma)} \left\lceil \frac{r_n + j_{\tau'}}{p_{\tau'}} \right\rceil c_{\tau'}^+$$

PROOF. See [79]. □

In [122] some improvements for a faster implementation and in [32] better initial start values are proposed. In [21] the response time of several consecutive jobs and in [91] for transactions of tasks is considered.

An alternative analysis is the scheduling point test [80]. A scheduling point S for a task τ is a point in time in which all existing jobs of tasks with equal or higher priority than τ and the task itself are finished. For the task itself it is only necessary that the currently considered job is finished. In this case, the schedulability with regard to a specific deadline

d_{τ_i} is given when there exists a scheduling point with regard to τ_i that is equal or smaller than d_{τ_i}.

THEOREM 2.1.6. *(similar to [80, 81]) A task set Γ is schedulable with respect to a task τ when there exists a time point $S \leq d_\tau$ with:*

$$S \leq \sum_{\forall \tau' \in hp(\tau,\Gamma) \cup \tau} \left\lceil \frac{S}{p_{\tau'}} \right\rceil c^+_{\tau'}$$

PROOF. See [80]. □

This test can be extended to the arbitrary case. Due to the possible existence of several concurrent instances of a task the first scheduling point might occur later than the deadlines of the first instances of a task. It is then necessary to additionally prove for all deadlines D occurring before the first scheduling point S that there exists a reduced scheduling point S' with regard to all tasks of higher priority and all such instances of the task that have to be finished at deadline D.

Manabe and Aoyagi [92] reduce the set of relevant scheduling points and proved that in all cases in which a scheduling point exists, at least one element of the reduced set is also a scheduling point.

THEOREM 2.1.7. *[92] If there exists a scheduling point with regard to a task τ than one point in R is also a scheduling point*

$$R = \bigcup_{\tau' \in hp(\tau,\Gamma) \cup \tau} Q^\tau_{\tau'}$$

$$Q^\tau_{\tau'} = \{ \lfloor \frac{t}{p_{\tau'}} \rfloor p_{\tau'} | t \in (\bigcup_{\tau'' \in hp(\tau,\Gamma) \setminus hp(\tau',\Gamma)} Q_{\tau''} \wedge t < \lfloor \frac{t}{p_{\tau'}} \rfloor p_{\tau'} + d_{\tau'}) \}$$

$$Q^\tau_\tau = \{ d_\tau \}$$

PROOF. See [92] □

Other than in previously introduced analysis concepts, the size of this set and therefore the complexity of this test only depends on the number of tasks in the task set and is independent of the periods and deadlines involved. The size of the set is exponential in the number of tasks and therefore the analysis has an exponential complexity too.

2.2. Schedulability analysis for task sets with dynamic priorities

Allowing a dynamic change of the priorities of tasks taking the concrete schedule into account can lead to a higher possible utilization of a processor and therefore to the schedulability of more tasks on the same processor. The most important scheduling rule using dynamic priorities is earliest deadline first scheduling (EDF). Liu and Layland proved in [88] that EDF is an optimal scheduling in the sense that if a task set is schedulable on a processor and keeps all its deadlines, it is schedulable using EDF scheduling.

2.2.1. Analysis with linear complexity. For the simple task and event model in which every task has a stimulation period being larger or equal to the deadline of the task, EDF allows a utilization of 100% (without consideration of the scheduling overhead). Therefore each task set that does not overload a resource is schedulable with EDF scheduling. Remember, that this bound is as low as 69.3% for fixed-priority scheduling.

The only step required for a schedulability analysis for EDF with this bound is calculating the utilization of the task set on its resource. As no task set that overloads a resource is schedulable this simple test is sufficient and necessary. Unfortunately, the situation is not that easy when using a more restricted task or event model. In the case that a task set includes at least one task having a deadline smaller than the period of its events, the utilization-based analysis is no longer sufficient.

There are simple sufficient schedulability analyses even for less restricted models available. One can be followed out of the utilization-based analysis by exchanging the period with the deadline of the task.

THEOREM 2.2.1. *[90] A task set Γ containing tasks $\tau \in \Gamma$ with deadline d_τ smaller than their period p_τ is schedulable on a given resource if*

$$\sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{d_\tau} \leq 1$$

PROOF. Assume a more restricted task set Γ' in which all tasks are assigned a period $p_{\tau'}$ with $p_{\tau'} = \min(d_\tau, p_\tau)$. For this task set the following condition holds: $\sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{d_\tau} \geq \sum_{\forall \tau' \in \Gamma'} \frac{c_{\tau'}^+}{p_{\tau'}} = U_{\Gamma'}$.

If $\sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{d_\tau} \leq 1$ then also $\sum_{\forall \tau' \in \Gamma'} \frac{c_{\tau'}^+}{p_{\tau'}} \leq 1$, which means that the task set Γ' is schedulable if the task set Γ fulfills the lemma. Γ' is only a restricted version of Γ and so Γ can be generated out of Γ' by relaxing some periods. As Γ' is schedulable and relaxing periods of a schedulable task set leads again to a schedulable task set, Γ is schedulable. □

Unfortunately, this test is only sufficient and many schedulable task sets cannot be recognized as schedulable by this test. The utilizations achievable by the recognized schedulable task sets are quite lower than the average utilizations of all schedulable task sets. For example, if a task set includes a task having a deadline equal to its worst-case execution time, the test would allow no other tasks in the task set even if this task only needs a low fraction of the available processor time. Therefore a more realistic analysis was required.

Devi improved the sufficient analysis in [46].

THEOREM 2.2.2. *[46] A task set Γ is schedulable if for each task $\tau \in \Gamma$ the following condition holds:*

$$\sum_{\substack{\forall \tau' \in \Gamma \\ d_{\tau'} \leq d_\tau}} \frac{c_{\tau'}^+}{p_{\tau'}} + \frac{1}{d_\tau} \sum_{\substack{\forall \tau' \in \Gamma \\ d_{\tau'} \leq d_\tau}} \left(\frac{p_{\tau'} - \min(p_{\tau'}, d_{\tau'})}{p_{\tau'}} \right) c_{\tau'}^+ \leq 1$$

PROOF. See [46] □

This test is more accurate than the previous test. It was also proved that this test returns "schedulable" in all those cases in which the previous test returns also "schedulable". Unfortunately, the test is still only sufficient. In section 3.7 we will investigate the sufficiency of the test and show interesting relationships of this test and results achieved in this theses.

Recently Masrur et al. proposed another sufficient test which we will introduce at the end of section 2.2.3 as it requires some more theory in advance.

For a sufficient and necessary proof of the schedulability of a task set having at least one task with a deadline smaller than the period of the task's events a more complex analysis is necessary. An inefficient approach is the transfer of the worst-case response-time analysis on dynamic priority scheduling approaches [47, 58, 125]. The processor demand criterion is much more efficient.

2.2.2. Processor Demand Criterion. The processor demand criterion was, in a simplified version, proposed by Leung & Merill [83] and improved by Baruah et al. [19] for the periodic task model. Later Gresser [60] proposed independently a similar test for a more advanced event model, the event streams. The problem is reduced to an efficient computation of one function; the demand bound function.

The idea is to calculate for each possible length of intervals Δt the maximum demand of computation time that has to be processed in any interval of this length in any possible schedule. This demand belongs to jobs having both, their invocation and their absolute deadline within the interval. This demand can be described by an abstract demand bound function.

DEFINITION 2.2.3. (Demand Bound Function δ) [13, 19, 60, 61, 83]
Let Γ be a simple periodic task set. The demand bound function $\delta(\Delta t, \Gamma)$ returns for each interval-length Δt the worst-case demand that has to be processed by all those jobs for which both, the release time and the deadline is included in Δt.

COROLLARY 2.2.4. *([83, 60]) The maximum demand bound function for any interval-length Δt is given by:*

$$\delta(\Delta t, \Gamma) = \sum_{\substack{\forall \tau \in \Gamma \\ \Delta t \geq d_\tau}} \left\lfloor \frac{\Delta t - d_\tau}{p_\tau} + 1 \right\rfloor c_\tau^+$$

PROOF. Let τ_τ be the set of jobs generated by task $\tau_i \in \Gamma$ and $\tau_{\tau,i} \in \tau_\tau$ be one of those jobs. Let $r_{\tau_{\tau,j}}$ be the release time of the job and $D_{\tau_{\tau,j}}$ be the absolute deadline of the job. Let there be a schedule for the tasks of Γ having a point of time t_0 in which one job of each τ_τ is released, so all tasks are released simultaneously in t_0. The demand bound function $\delta(\Delta t, \Gamma)$ consists of the sum of the worst-case execution times of all those jobs having a release-time $r_{\tau_{\tau,i}} \geq t_0$ and an absolute deadline $D_{\tau_{\tau,i}} \leq t_0 + \Delta t$. The number of these jobs for a task τ is given by $\left\lfloor \frac{\Delta t - d_\tau}{p_\tau} + 1 \right\rfloor$ if $\Delta t \geq d_\tau$.

The condition $\Delta t \geq d_\tau$ prevents negative numbers of events. The first job being fully included into an interval is the first job that starts at the begin of the interval and ends at its deadline. So the minimum interval including one job of τ has the length d_τ and is also the minimum interval for which the calculation returns one. The second job of τ starts at the

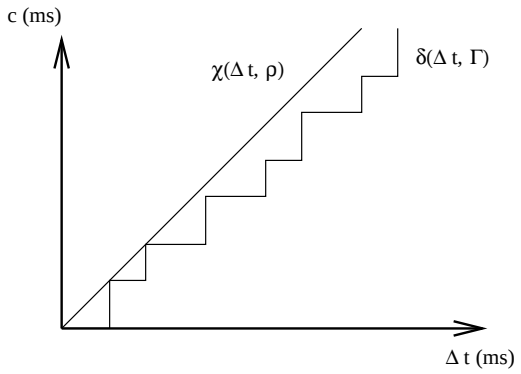

FIGURE 2.2.1. Example demand bound function

time point $0 + p_\tau$ and ends at $p_\tau + d_\tau$. The calculation returns two for the interval of length $p_\tau + d_\tau$, and so on. So, the calculation returns the exact number of those jobs of a task occurring within Δt for the case in which the start point of the first job and of the interval is the same.

Obviously, a job that is released at or after the start of an interval and has to be finished at or before the end of this interval needs to be processed completely within this interval. Therefore the job needs, in the worst case, as much processing time within the interval as it has as worst-case execution time. As it is allowed to release all tasks simultaneously the schedule calculated by the lemma can occur. So, it only remains to prove that this schedule leads directly to the worst-case value for the demand bound function and that there exists no other schedule leading to more demand. Such a schedule would require that for at least one task more jobs are released and finished within Δt as with the schedule above. As the first release of each job is exactly at the start of the interval that would mean that postponing this release could lead to more jobs. That is not possible. Note, that we consider each interval-length separately, so we have not to consider any job with either release time or deadline outside of Δt. Therefore the intervals starting at the simultaneously release of jobs of all tasks describes the worst-case contributions for the demand bound function. □

EXAMPLE 2.2.5. Consider the task set $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ with $\tau = (p_\tau, c_\tau^+, d_\tau)$ and $\tau_1 = (8\,ms, 4\,ms, 4\,ms)$, $\tau_2 = (22\,ms, 3\,ms, 7\,ms)$, $\tau_3 = (19\,ms, 3\,ms, 17\,ms)$, $\tau_4 = (30\,ms, 1\,ms, 26\,ms)$. The task set is visualized in figure 2.0.1. The demand bound function of this task set is visualized in figure 2.2.1.

For the simple period task model the worst-case demand for any interval Δt is given by one worst-case schedule of the task set. This is not a necessary condition for the demand bound function, as we will see later. The worst-case situations for different intervals can result out of different worst-case schedules. The demand bound function is than a combination of all worst-case schedules.

The idea behind the processor demand test is to prove for intervals of each possible length that the maximum demand of computation time can be satisfied by the available computation time within the interval.

The available computation time depends on the resource ρ on which the task set is mapped. It can be described by a function similar to the demand bound function.

DEFINITION 2.2.6. Capacity bound function

The capacity bound function $\chi(\Delta t, \rho)$ returns for each interval-length Δt the minimum amount of computation time that is available for the execution of tasks on a resource ρ within any interval of length Δt.

The given worst-case execution time is measured on an idealized processor with $\chi(\Delta t) = \Delta t$ for all interval-lengths Δt. For simplicity, $\chi(\Delta t) = \Delta t$ is considered as capacity bound function by the processor demand test.

THEOREM 2.2.7. *Processor Demand Test*

A task set Γ is schedulable if and only if for all possible interval-lengths I the demand bound function $\delta(\Delta t, \Gamma)$ is smaller or equal than the amount of execution time available within Δt on the resource ρ. ρ is the resource on which Γ is executed:

$$\delta(\Delta t, \Gamma) \leq \chi(\Delta t, \rho)$$

So, in the simplified case we have to check $\delta(\Delta t, \Gamma) \leq \Delta t$. For the example of above the test is visualized again in figure 2.2.1.

PROOF. The proof follows [60]. Let us assume that the condition holds and that there is at least one deadline miss. Let us consider a reduced schedule S' eliminating all jobs with a deadline later than the missed deadline. We consider the demand bound function for an interval Δt that ends at the point of time where the deadline is missed and starts at the last idle time of the reduced schedule S' before the missed deadline. There exists an idle time at least at the origin. The existence of an idle time requires that at this time there are no jobs available for processing. Therefore all jobs being processed within Δt are also released within Δt. As we have no tasks within the reduced schedule S' having a deadline later than the end of Δt, all jobs being executed within Δt contribute to the demand bound function. If a deadline is missed within Δt at least one time unit of the processing time required by the job that misses the deadline is not processed within Δt. There are only two possibilities to reach such an additional processing time of τ after its deadline. First, the jobs require more processing time than available within Δt. In this case the demand bound function would exceed the capacity bound function for this interval, which is in contradiction to the assumption. Second, there exists at least one time unit within Δt in which the processor is either idle or a task is executed which does not belong to S'. That means that there exists an idle time within Δt in relation to the reduced schedule S'. But this is in contradiction to the definition of Δt. □

The question is which intervals have to be considered by the processor demand test. These are all those intervals for which the value of the demand bound function can change. The concrete values depend on the task set and the chosen event model. For the periodic task model these values are all those interval sizes that match exactly with the minimum difference between the occurrence times of several jobs of the same task. Collecting these

times for all tasks and all number of instances gives the set of test intervals for the processor demand criterion. For the periodic task model the set for one task τ is given by

$$\Delta t = kp_\tau + d_\tau$$

with $k \in \mathbb{N}_0$. The union of all sets of all tasks gives the complete set.

The problem of the test is reduced to an efficient computation of the demand bound function. In this version the processor demand criterion would require to check all intervals up to an infinite size.

2.2.3. Maximum test interval. To make the test feasible it is necessary to limit the maximum considered interval. To give a complete overview we will briefly summarize in the following the results and the proofs for the results achieved so far in literature. We will start with a definition of the term "maximum test interval".

DEFINITION 2.2.8. *Maximum test interval Δt_{max}*

Let Γ be a task set with $U_\Gamma \leq 1$. A maximum test interval is an interval-length Δt_{max} so that in case of the existence of an interval-length Δt with $\delta(\Delta t, \Gamma) > \chi(\Delta t, \rho)$ and $\Delta t > \Delta t_{max}$ there exists also an interval $\Delta t' \leq \Delta t_{max}$ with $\delta(\Delta t', \Gamma) > \chi(\Delta t', \rho)$.

For the periodic task model there are several maximum test intervals available.

The first one is the least common multiple of the event-periods of all tasks.

THEOREM 2.2.9. *[83] The least common multiplier (LCM) of the event-periods of all tasks in the task set is a maximum test interval.*

PROOF. By definition the first event of task τ occurs at time-point zero. As the pattern of the following events are given by their periods, the following event occurs at time p_τ, the next at time $2 \cdot p_\tau$ and so on. So one event of a task τ occurs at every multiple of its period p_τ. Therefore an event of the task occurs at the least common multiplier of all periods of all tasks. Therefore an event of each task occurs at this point of time.

The request of execution C_{LCM} within Δt_{LCM} is given by $C_{LCM} = \sum_{\forall \tau \in \Gamma} \left\lceil \frac{\Delta t_{LCM}}{p_\tau} \right\rceil c_\tau^+$. As $\left\lceil \frac{a}{b} \right\rceil = \left(\frac{a}{b}\right)$ if b divides a,

$$\sum_{\forall \tau \in \Gamma} \left\lceil \frac{\Delta t_{LCM}}{p_\tau} \right\rceil c_\tau^+ = \sum_{\forall \tau \in \Gamma} \left(\frac{\Delta t_{LCM}}{p_\tau} \right) c_\tau^+ = \Delta t_{LCM} \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau}$$

.

As $\sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau} \leq 1$ there has to be an end of the first busy period within Δt_{LCM} and therefore if there exists a miss of a deadline somewhere in the schedule, one miss exist within Δt_{LCM}. □

The disadvantage to use the least common multiple as maximum test interval is that its length heavily depends on the concrete values of the periods. In the case that the periods are neither integer-values nor multiples of one common step-size it can be impossible to find an LCM smaller than the product of all periods. For event models with offsets the maximum offset has to be added to the LCM for the maximum test interval.

Another maximum test interval is the general busy period condition, which is also not limited to the simple periodic task model.

DEFINITION 2.2.10. [119] *Busy period*

The busy period is the length of the maximum interval which does not include an idle point. An idle point is a point of time in which no job is ready to be processed, e.g. the processing for each job for which the activating event has occurred is already finished. The idle point can be an infinitely small moment of time if the completion of the last available job and the invocation of the first new job happen at the same point of time.

THEOREM 2.2.11. *[119] Busy period condition*

The busy period of a task set in which the first events of all tasks occur simultaneous at the start of the busy period is a maximum test interval for each kind of task set.

PROOF. We have to proof that in those cases in which a failure of a deadline occurs, a failure occurs within the first busy period. Let us assume the contrary, e.g. that the first failure occurs in one of the following busy periods. Other than for the first busy period the first events of the task do not have to occur simultaneously in the following busy periods. They can occur with any pattern. Only events arriving within the busy period can contribute to the execution time that has to be processed within this busy period. This follows out of the idle point that has to occur by definition right before the start of the busy period and prevents the existence of any unfinished job. Therefore each busy period can be considered separately, only distinguished by their event pattern. The worst-case pattern is the simultaneous release of events of all tasks, which is the pattern for the first busy period. So if any pattern leads to a missed deadline, the synchronous release would lead to such a miss too, so a miss would have to occur within the first busy period. □

For the periodic task model the busy period is the smallest value of Δt for which the following condition holds:

$$\mathscr{B}(\Gamma) = \min\left(\Delta t \,|\, \Delta t \geq \sum_{\forall \tau \in \Gamma} \left\lceil \frac{\Delta t}{p_\tau} \right\rceil c_\tau^+\right)$$

The disadvantage to calculate the maximum test interval by this formula is that a fix-point iteration is required which can result in many calculations to achieve the result.

There are some other maximum test intervals available so far. The first one was developed 1990 by Baruah et al. [19]:

THEOREM 2.2.12. *[19] The interval*

$$\Delta t_{max} = \max_{\forall \tau \in \Gamma}(p_\tau - d_\tau)\frac{U_\Gamma}{1-U_\Gamma}$$

with $U_\Gamma = \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau}$ *is a maximum test interval for the task set* Γ

Later Ripoll et al. [119] showed another maximum test interval leading to a stricter bound:

THEOREM 2.2.13. *[119] The interval:*

$$\Delta t_{max} = \frac{\sum_{\forall \tau \in \Gamma}\left(1 - \frac{d_\tau}{p_\tau}\right)c_\tau^+}{1-U_\Gamma}$$

is a maximum test interval for the task set Γ *if* $\forall \tau \in \Gamma : d_\tau \leq p_\tau$.

If a task set is not feasible, a deadline is missed before the maximum test interval of Ripoll et al. The proofs for these two maximum test intervals can be done together:

PROOF. (Following the proof in [19]) Let Δt be the interval for which the analysis fails, e.g. the demand bound function exceeds the intersection, which models the available computation time. That means Δt fulfills the following conditions:

$$\Delta t < \sum_{\forall \tau \in \Gamma} \left\lfloor \frac{\Delta t - d_\tau}{p_\tau} + 1 \right\rfloor c_\tau^+$$

$$\Delta t < \sum_{\forall \tau \in \Gamma} \left(\frac{\Delta t - d_\tau}{p_\tau} + 1 \right) c_\tau^+$$

$$\Delta t < \sum_{\forall \tau \in \Gamma} \left(\frac{\Delta t - d_\tau + p_\tau}{p_\tau} \right) c_\tau^+$$

$$\Delta t < \Delta t \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau} + \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau} (p_\tau - d_\tau)$$

$$\Delta t < \frac{\sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau} (p_\tau - d_\tau)}{1 - \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau}}$$

This result can easily be transferred into the maximum test interval of Ripoll et al. [119]:

$$\Delta t < \frac{\sum_{\forall \tau \in \Gamma} \left(1 - \frac{d_\tau}{p_\tau}\right) c_\tau^+}{1 - U_\Gamma}$$

So, the demand bound function can only exceed the intersection for values smaller or equal to Δt.

The goal for Baruah et al. [19] was to prove the complexity of the analysis. Therefore they used an upper bound of the above in-equation depending mainly on the utilization:

$$\Delta t < \frac{\sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau} (p_\tau - d_\tau)}{1 - \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau}}$$

$$\Delta t < \frac{\sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau} \max_{\forall \tau' \in \Gamma} (p_{\tau'} - d_{\tau'})}{1 - \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau}}$$

$$\Delta t < \frac{U_\Gamma}{1 - U_\Gamma} \max_{\forall \tau \in \Gamma} (p_\tau - d_\tau)$$

□

George et al. [55] extended the maximum test interval of Ripoll et al. [119] to task sets with tasks having deadlines larger than their periods.

THEOREM 2.2.14. *[55] The interval*

$$\Delta t_{max} = \frac{\sum_{\forall \tau \in \Gamma} \left(1 - \frac{min(d_\tau, p_\tau)}{p_\tau}\right) c_\tau^+}{1 - U_\Gamma}$$

is a maximum test interval for the task set Γ

PROOF. The proof is quite similar to the proof above but now the condition $\forall \tau \in \Gamma : d_\tau \leq p_\tau$ is not required any more.

$$\Delta t < \sum_{\substack{\forall \tau \in \Gamma \\ \Delta t > d_\tau - p_\tau}} \left\lfloor \frac{\Delta t - d_\tau}{p_\tau} + 1 \right\rfloor c_\tau^+$$

$$\Delta t < \sum_{\forall \tau \in \Gamma} \left\lfloor \frac{\Delta t - min(d_\tau, p_\tau)}{p_\tau} + 1 \right\rfloor c_\tau^+$$

$$\Delta t < \frac{\sum_{\forall \tau \in \Gamma} \left(1 - \frac{min(d_\tau, p_\tau)}{p_\tau}\right) c_\tau^+}{1 - U_\Gamma}$$

□

Park and Cho proposed in [103] another maximum test interval.

THEOREM 2.2.15. *[103] The interval*

$$\Delta t_{max} = \min\left(\Delta t | \Delta t \geq \sum_{\forall \tau \in \Gamma} \left(\left\lfloor \frac{\Delta t}{p_\tau} \right\rfloor c_\tau^+ + \frac{c_\tau^+}{d_\tau} \min\left(d_\tau, \Delta t \mod p_\tau\right)\right)\right)$$

is a maximum test interval for the task set Γ.

PROOF. See [103] □

The problem with this bound is that the condition for the maximum test interval has to be calculated for every interval Δt separately. This leads to a high effort during the analysis.

Recently Masrur et al. [93] improved the maximum test interval of Ripoll et al. by showing that the interval itself can be excluded, so the first deadline miss has to happen before this bound. This conclusion is applicable to all the maximum test intervals described before.

THEOREM 2.2.16. *(Similar to [93]) For a non-feasible task set a deadline is missed before the maximum test interval is reached.*

PROOF. Let Δt be the smallest interval leading to a deadline miss. Therefore the available computation time for Δt is not sufficient to cover the requested demand ($\Delta t < \delta(\Delta t, \Gamma)$). As already shown in the proofs of above this leads to an interval really smaller than the maximum test intervals. □

Masrur and Färber proposed in [94] a bound quite similar to the bound proposed by Ripoll et al. [119]. For improving this bound, Masrur and Färber use the fact that an infeasible task set leads to a deadline-miss in size of at least one time unit. They are considering the inverse synchronous schedule (e.g. the schedule build in a way that it contains a job of each task with a deadline at the end of the hyper-period). In case that a task set is not feasible and has an utilization equal or lower to 100%, at least one deadline is missed at the end of the hyper-period when using the inverse synchronous schedule. Therefore an additional idle time with size of at least one time unit exists within this hyper-period. It exists additional to the idle time that occurs inevitably for task sets having a utilization lower than 100%.

THEOREM 2.2.17. *(Similar to [94]) A task set Γ is feasible if and only if the requested processor demand for all tasks in the inverse synchronous schedule is larger than the available computation time for all intervals smaller or equal the maximum test interval Δt_{MF} (using a discrete time scale):*

$$\Delta t_{max} = \frac{\sum_{\tau\in\Gamma}(1-\frac{d_\tau}{p_\tau})c_\tau^+ - 1}{1-U_\Gamma}$$

PROOF. See [94] □

To prove schedulability using this new maximum test interval Δt_{max} Masrur and Färber propose to complete the task set with a task covering the utilization gap between the utilization of the task set itself and 100% utilization and the check for every possible interval up Δt_{MF} if an idle time can occur.

The bound is that it requires a discrete time scale, so each deadline, period and worst-case execution time is a multiple of a common time unit. It is also not suitable for approximative analysis, as for finding the additional idle times it is necessary to close the gap between the utilization of the task set and a utilization of 100% leaving no space for approximation.

Masrur et al. [93] extended this result also to a new sufficient feasibility test. The test is partly based on the relationship between the test of Devi [46] and the test bound of George et al. [55] as indirectly shown in [6]. The idea is to calculate a maximum test interval separately for each task based only on the task in question and all tasks having an equal or smaller deadline than the task in question. In case that for each task the maximum test interval is equal or smaller than the deadline of the task the schedulability is proved.

THEOREM 2.2.18. *(Generalization of [93]) Let Δt_{max}^τ be a test interval based only on the task $\tau\in\Gamma$ and all tasks in $\Gamma_\tau = \{\tau'\in\Gamma | d_{\tau'} \le d_\tau\}$. The schedulability is given if $\Delta t_{max}^\tau \le d_\tau$ holds for all $\tau\in\Gamma$.*

PROOF. As we know from [88] the simultaneous release of all tasks is the worst-case situation. So due to the EDF scheduling scheme a task cannot be postponed by a task with a larger deadline so these tasks can be ignored completely. As a result we can use $\tau\cup\Gamma_\tau$ for calculating Δt_{max}. From definition 2.2.8 we know that a non-feasible task set has a deadline miss within the maximum test interval and therefore also before the deadline of τ. □

Using the maximum test interval of George et al. [55] and this theorem results in the test of Devi [46]. Masrur et al. used a new, tighter, maximum test interval which is an improvement of theorem 2.2.17.

THEOREM 2.2.19. *[93] Let there be a task set Γ with n tasks $\tau\in\Gamma$ sorted by their increasing deadline, so $i<j \Rightarrow d_{\tau_i} \le d_{\tau_j}$. The interval Δt_{max}^k is a maximum test interval with*

$$\Delta t_{max}^k = \frac{\sum_{i=1}^k x_i^{(i-1)} c_{\tau_i}^+ + \sum_{j=k+1}^n (p_{\tau_j} - min(p_{\tau_i}, d_{\tau_j}))U_{\tau_j}}{1-U_\Gamma+\sum_{i=1}^k U_{\tau_i}}$$

with

$$x_i^j = max\left(0, \left\lceil \frac{\Delta t_{max}^j - d_{\tau_i}}{p_{\tau_i}} \right\rceil\right)$$

and $\Delta t_{max}^0 = \Delta t_{max}$ *as the maximum test interval of theorem 2.2.14.*

PROOF. The proof is given in [93]. Starting point is again that for an infeasible task set the deadline miss has to happen before the maximum test interval of theorem 2.2.14. For this maximum test interval Δt_{max} the maximum effort for a concrete task can be bounded by the exact number of jobs x_i^j processed at most within the maximum test interval. □

In chapter 6 we have compared the execution times for the schedulability analysis for all these different maximum test intervals for a huge set of randomly generated task sets and compared them with our results.

2.2.4. The Problem of complexity. Let us now consider the complexity of the processor demand criterion, which will give a motivation for this thesis. It depends on the effort to calculate and evaluate the demand bound function for one interval and on the number of those intervals, test intervals, being necessary to be tested to decide the problem of feasibility. See [16] for an overview of the previous results on complexity analysis.

For feasibility analysis we need to consider the worst-case test intervals of each possible length. From the definition of the demand bound function we know that each test interval starting at the origin of the demand bound function is the worst-case test interval of its specific length. Each test interval requires the evaluation of the demand bound function for one value only. The complexity for the evaluation is $O(n)$ where $n = |\Gamma|$ is the number of tasks τ in the task set Γ. The problem is that the number of test intervals does not only depend on the number of tasks in a task set. The number of test intervals is determined by the maximum test interval on the one side and the maximum density of test intervals on the other side.

The length of the maximum test intervals depends mainly on the utilization of the task sets. The maximum test intervals of Ripoll et al. [119] and of Baruah et al. [19] depend directly on the utilization. The length of the busy period, that covers the interval up to the first idle point of the system, depends on how often such an idle point or such an idle time occurs (An idle point is defined as a point in time at which all available jobs are finished; only jobs arriving at exactly this point of time are allowed to exist. It can be regarded as an idle time with length zero). The density of idle points respectively idle times (and their length) is closely related to the utilization. A system with 50% utilization has much more idle times than one with 99% utilization. The distance to the first idle point is normally smaller in systems with 50% utilization than in those with 99% utilization.

The test interval of lemma 2.2.9 is independent of the utilization value. It depends only on the tasks of the task set. But this test interval is normally much larger than the previously considered test intervals. It cannot be bound using the utilization. In the worst case the "least common multiple" is equal to the product of the periods of all tasks. Therefore we can bound its length only with an exponential value ($max(p_\tau)^{|\Gamma|}$).

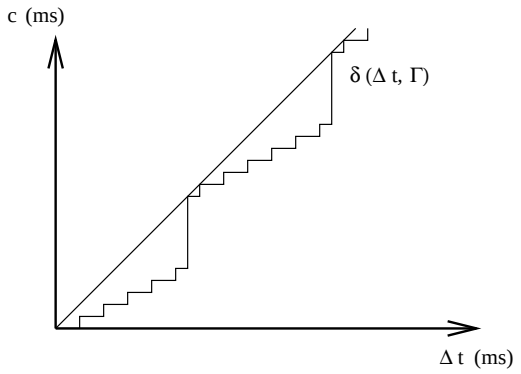

FIGURE 2.2.2. Example demand bound function with large ratio

The problem is that a task set can contain tasks of very different sizes. The size of the task is determined by the value of all parameters of the task, its period, its deadline and its worst-case execution time. Consider for example a task τ_a with $p_{\tau_a} = 10$, $d_{\tau_a} = 8\,ms$ and $c^+_{\tau_a} = 3\,ms$ and a task τ_b with $p_{\tau_b} = 1000\,ms$, $d_{\tau_b} = 800\,ms$ and $c^+_{\tau_b} = 300\,ms$. Both tasks lead to the same utilization of 30% and the hardness of schedulability is also the same for both tasks. But they have different sizes, as τ_b is 100 times larger than τ_a.

It is possible to have very large tasks and very small tasks in one single task set. The maximum test interval depends somehow on the largest task or on the average size of the tasks. For example the value of the maximum test interval of Baruah et al. [19] depends on the maximum of the differences between the period and the deadline of any task of the task set ($max_{\forall\tau\in\Gamma}(p_\tau - d_\tau)$). The maximum test interval of Ripoll et al. [119] depends on the ratio between the deadline and the period of the task but also on the worst-case execution time and therefore on the size of the task itself. In contrary to that, the density of the test intervals depends on the smallest tasks of the task set.

The problem is that the maximum test interval depends on the whole task set. But every task of the task set, even the smallest, requires a set of test intervals that has to cover the complete length of the maximum test interval. So, a task set having tasks with a large size and therefore a large maximum test interval and at least one very small task would need a large number of test intervals for this task. This number will dominate the overall number of test intervals. For example, having a maximum test interval of 100 ms and a task with a period of 20 ms would lead to no more than five test intervals for this task whereas having a maximum test interval of 100 s can lead to up to 5000 test intervals for the same task. A demand bound function for such a task set is visualized in figure 2.2.2.

A larger ratio between the sizes of the tasks in one task set leads to a larger effort for the schedulability analysis. Its complexity depends on the ratio of the sizes and therefore on the values of the parameters of the task.

An algorithm has a pseudo-polynomial complexity if it runs in polynomial time in case that the parameters are represented by unary numbers but does not run in polynomial time if the parameters are represented by binary numbers. So, the complexity for pseudo-polynomial algorithms does not depend on the value of one single parameter but on the

values of all parameters. In our example the complexity does not depend only on the number of tasks but also on the size of the task. The size is represented by the period, the deadline and the worst-case execution time of the task. For an algorithm with polynomial complexity it can depend on one parameter only, for example on the number of tasks in the task set.

If the utilization is bounded by a value $U_{max} < 1$, the processor demand criteria, in the form we have introduced so far, has a worst-case complexity of $O(n^2 \frac{max_{\forall \tau \in \Gamma}(max(p_\tau, d_\tau)}{min_{\forall \tau \in \Gamma}(min(p_\tau, d_\tau))})$ where n is the number of tasks in the task set Γ. It is sufficient to use only the maximum respectively minimum periods of all tasks and ignore the deadlines and the worst-case execution times. If its utilization value is bounded, the processor demand analysis has a pseudo-polynomial complexity.

For a general schedulability analysis the dependency of the complexity on the parameters of the tasks and especially on the ratio between different tasks is a problem. The run time of the analysis becomes unpredictable. The problem has also a practical implication. Such task sets with a large ratio occur in many real-world applications. For example consider an embedded system with a small real-time operating system. The tasks of the operating system like task switching and the scheduling decision functions as well as polling requests for sensor data can have run-times in the area of nanoseconds or less whereas complex tasks for the evaluation of the sensor data or complex decision algorithms can have run-times in the area of seconds or minutes.

We will propose in chapter 3 and chapter 4 solutions for this problem.

2.3. Event models

After the introduction of different schedulability analysis methods we will now consider the existing possibilities to describe the stimuli for the tasks.

We distinguish between tasks models and event models, which separates the stimuli from the other parameters of the tasks.

A huge set of specialized task models have been proposed so far. Examples for advanced task models are the multi-frame task model [41, 96] and its extension the generalized multi-frame task model [14]. These models allow the variation of the execution time, the deadlines and the periods of the tasks. The model consists of a periodically repeated chain of tasks with a fixed order where each task is associated with separate values for its parameters. For the analysis it is necessary to build step-wise the worst-case situations. The recurring real-time task model [12, 13] generalizes this approach by allowing branches in the chain.

In literature several event models relaxing the strictly period stimuli model of Liu and Layland [88] where proposed. The first one was the introduction of sporadic tasks, having a minimum inter-arrival time instead of the period. The distance between events of sporadic tasks may be longer than their minimum inter-arrival time, but not shorter.

We are using the same symbol p for both, the period of strictly periodic tasks and the minimum inter-arrival time of sporadic tasks. In the periodic task model the events have to occur exactly at the period whereas in the sporadic task model the events are allowed

to occur later than the time given by their periods. There has to be at least the distance of one period-length to the next following event, so the lateness of one event will lead to the lateness of all following events. The previously introduced schedulability analyses are applicable to both models.

Widely accepted is the introduction of a jitter interval. The occurrences of the events are no longer strictly periodic. Instead, around the place of the expected occurrence of an event a jitter interval j is defined and the event is allowed to occur anywhere within j. The interval is normally defined in a way that the occurrence of the event in the strictly period case would be in the middle of the jitter interval so the event is allowed to occur earlier or later by the same amount of time. Since each event is allowed to jitter in the same way, the minimum distance between two events becomes $p - j$ and the maximum distance (for strictly periodic tasks) becomes $p + j$.

2.3.1. Periodic task model. We will reconsider the periodic task model first. This model was originally described in the seminal work of Liu and Layland [88]. In this work the tasks of one processor are modeled by a task set Γ consisting of independent tasks τ. One task can be invoked continuously. Each invocation of a task is called a job. The i-th job of the n-th task is denoted $\tau_{n,i}$. The workload of the task is bounded by the worst-case execution time c^+, the available time to process this workload is bounded by a relative deadline d. A system in which the execution of a job has to be finished within his deadline is called a hard real-time system. In the model the same relative deadline is assigned to each job of a task. The absolute deadline of a job is given by the time of invocation of the job and its relative deadline. It describes the point of time at which the execution of a task has to be finished.

In the periodic task model each task τ of a task set Γ is assigned a period p, a deadline d and a worst-case execution time c^+ and, optionally, a maximum jitter interval j. This is sufficient for strictly periodic stimuli. Sensors monitoring the environment and doing measurements with a constant rate can for example generate such a kind of stimuli. Sporadic tasks can be modeled by the periodic task model sufficiently by setting the period equal to the minimum distance between either two invocations of the task.

2.3.2. Event Streams. Event streams where first defined by Gresser [60]. In the following we will give our own introduction of the event-stream model. The purpose of the event-stream model was to give a generalized description for stimuli. The basic idea is to provide an efficient general notation for the event bound function. The event-streams are a general model that can describe each kind of stimuli exactly. It is therefore an extension of the previously described periodic model. This model is mainly based on one function; the event bound function.

For every interval Δt the event bound function $\eta(I)$ calculates the maximum number of those events which can occur within I. For this function only the length of I is relevant, not a specific begin and end point. In the following, the term interval refers to its length only.

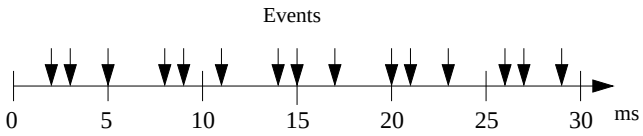

FIGURE 2.3.1. Example Event Sequence

The goal of the event stream model is to provide an efficient general notation for this event bound function η. Let us start with a general notation for sets of events.

DEFINITION 2.3.1. *Event sequence* Θ

An event sequence Θ *is a set of events being notated by their distance to a common origin of time.*

The distance can also be regarded as a time interval between the start time and the event. We will call this set an event sequence. Let, for example, the set Θ = {2 ms, 3 ms, 5 ms, 8 ms, 9 ms, 11 ms, 14 ms, 15 ms, 17 ms, 20 ms, 21 ms, 23 ms, 26 ms, 27 ms, 29 ms} be an event sequence. The first event of the sequence arrives after 2 ms, the second 1 ms later and the third event additionally 2 ms later. The graphical representation for this small event sequence is given in figure 2.3.1.

Note that it is not possible to notate an infinite number of events in this way. Therefore events are grouped into periodic sequences. Such a periodic sequence can be modeled by a single tuple consisting of a period and an interval. The interval describes the distance for the first event of the sequence to the common origin.

DEFINITION 2.3.2. *Event element* θ

An event element $\theta = (p,a)$ *defines a set of periodic events. It consists of a period* p *and an initial interval (or offset)* a*. Each value* $k = a + np$ $(n \in \mathbb{N}_0)$ *describes an event of the set having* k *as the distance of the event to the common origin of time. A periodic event sequence* Θ *is a set of event elements* θ *and the union of the events described by each* $\theta \in \Theta$ *is the set of events belonging to* $\hat{\Theta}$*.*

An event tuple modeling only a single event can be notated using an infinite period ($p = \infty$), if it has no offset it can be notated as $(\infty, 0)$. Each possible event sequence can be described by a set of event elements. The example above can be notated as Θ={$(\infty s, 2ms)$, $(\infty s, 3ms)$, $(\infty s, 5ms)$, $(\infty s, 8ms)$, $(\infty s, 9ms)$, $(\infty s, 11ms)$, $(\infty s, 14ms)$, $(\infty s, 15ms)$, $(\infty s, 17ms)$, $(\infty s, 20ms)$, $(\infty s, 21ms)$, $(\infty s, 23ms)$, $(\infty s, 26ms)$, $(\infty s, 27ms)$, $(\infty s, 29ms)$}

The set can include the same event element several times. That would mean that several events could occur at exactly the same time.

In case that the event pattern of Θ is continued for an infinite time the event sequence can be notated in a much shorter way:

$$\Theta_2 = \{(6ms, 2ms), (6ms, 3ms), (6ms, 5ms)\}$$

The periodic event sequence consists of three event elements with offsets of 2 ms, 3*ms* and 5 ms and a common period of 6 ms. The first event element expands to 2 ms, 8 ms,

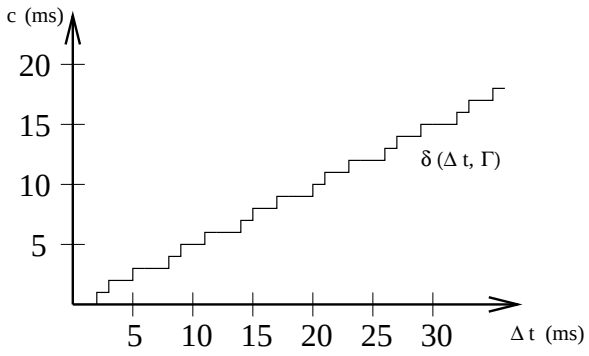

FIGURE 2.3.2. Example event bound function

14 ms, 20 ms, 26 ms, 32 ms, ..., the second to 3 ms, 9 ms, 15 ms, 21 ms, 27 ms, 33 ms, ... and the third to 5 ms, 11 ms, 17 ms, 23 ms, 29 ms, 35 ms, ... Together they form the infinite extension of the sequence Θ. This possibility to note infinite event sequences shortly leads to an efficient schedulability analysis.

The periodic event sequences are an extension of the periodic task model. The stimuli for a task τ with a period of 10 ms can be modeled by the periodic event sequence $\Theta = \{(10\,ms, 0\,ms)\}$. The stimuli for every task set described with the periodic task model can be also described exactly with one periodic event sequence for each task of the task set containing one event element.

DEFINITION 2.3.3. *Event Bound Function*

An event bound function provides the number of events occurring within the interval Δt located at the start of the sequence.

COROLLARY 2.3.4. *The event bound function $\eta(\Delta t, \Theta)$ for a periodic event sequence Θ and an interval I is given by:*

$$\eta(\Delta t, \Theta) = \sum_{\substack{\theta \in \Theta \\ \Delta t \geq a_\theta}} \left\lfloor \frac{\Delta t - a_\theta}{p_\theta} + 1 \right\rfloor$$

PROOF. Follows directly out of the definition of the periodic event sequence. □

The event bound function for the periodic event sequence Θ_2 is depicted in figure 2.3.2. The event bound function always has a monotonic non-decreasing behavior. A periodic event sequence is called homogeneous if all event elements share the same period or have an infinite period.

DEFINITION 2.3.5. *Homogeneous periodic event sequence*

A homogeneous periodic event sequence having a common period p is a periodic event sequence in which all event elements either share this common period p or have an infinite period.

For example $\Theta_3 = \{(\infty s, 1\,ms), (\infty s, 2\,ms), (10\,ms, 5\,ms), (10\,ms, 7\,ms)\}$ is such a homogeneous periodic event sequence. The periods of the sequence are either ∞ or have 10 ms as common value. A homogeneous periodic event sequence consists of two parts,

an aperiodic part containing all event elements with ∞ as period and a periodic part containing the other event elements. It is possible to transfer each periodic event sequence into a homogeneous one by exchanging the periods within the sequence by the least common multiple of the periods. To compensate this step it is necessary to insert additional event elements.

EXAMPLE 2.3.6. Consider the inhomogeneous periodic event sequence $\Theta_4 = \{(10\,ms, 3\,ms), (15\,ms, 7\,ms)\}$. The corresponding homogeneous periodic event sequence has 30 ms as the common period and needs therefore additional elements. For the first event element of the original sequence $(10\,ms, 3\,ms)$ three elements are required in the homogeneous sequence to generate the same set of events. These are the elements $(30\,ms, 3\,ms)$, $(30\,ms, 13\,ms)$, $(30\,ms, 23\,ms)$. For the second event element $(15\,ms, 7\,ms)$ two elements $(30\,ms, 7\,ms)$, $(30\,ms, 22\,ms)$ are required. The complete homogeneous periodic event sequence is given by Θ'_4 ={$(30\,ms, 3\,ms)$, $(30\,ms, 7\,ms)$, $(30\,ms, 13\,ms)$, $(30\,ms, 22\,ms)$, $(30\,ms, 23\,ms)$}.

In general a periodic event sequence Θ with p as the LCM of all periods (except ∞) can be transferred to a homogeneous periodic event sequence Θ' with
$\Theta' = \bigcup_{\theta \in \Theta'} \bigcup_{kp_\theta < p} (p, kp_\theta + a_\theta)$ and $k \in \mathbb{N}_0$.

For real-time analysis it is necessary that the event bound function gives for each interval the worst-case density of events. These densities need no longer belong to one concrete schedule only. They are the summary of the worst-case densities of all possible schedules. For the event bound function it is not necessary that there exist one single schedule containing the worst case for every possible interval. The worst-case densities can result from different scenarios and concrete time intervals. To model this event bound function for real-time analysis it is necessary to define an abstract event sequence modeling this cumulated worst-case densities, the event stream.

DEFINITION 2.3.7. *Event Stream*
An event sequence Θ is called event stream if for all intervals $\Delta t, J$:

$$\eta(\Delta t + J, \Theta) \leq \eta(\Delta t, \Theta) + \eta(J, \Theta)$$

The event stream is an event sequence in which the highest density of events occurs always at the start of the sequence. Together with the event bound function the event stream gives for each possible interval length the worst-case density of those events that can occur anywhere in any possible schedule of the task set.

In figure 2.3.3 some examples for event streams can be found. The first one is the event stream Θ_5 having a strictly periodic stimulus with a period p. The event stream can be notated as $\Theta_5 = \{(p, 0)\}$. So event streams having one event element only can model the event pattern activating a task of the simple periodic task model. In the second example Θ_6 a periodic stimulus in which the single events can jitter within a jitter interval of size j is depicted. $\Theta_6 = \{(\infty, 0), (p, p - j)\}$ is a description for this event stream. The third example Θ_7 is an event stream with an irregular behavior, three events occurring at the same time and the fourth occurring after a time t. This pattern is repeated with a period of p. The notation is Θ_7 = {$(p, 0)$, $(p, 0)$, $(p, 0)$, (p, t) } or shorter $\Theta_7 = \{3(p, 0), (p, t)\}$.

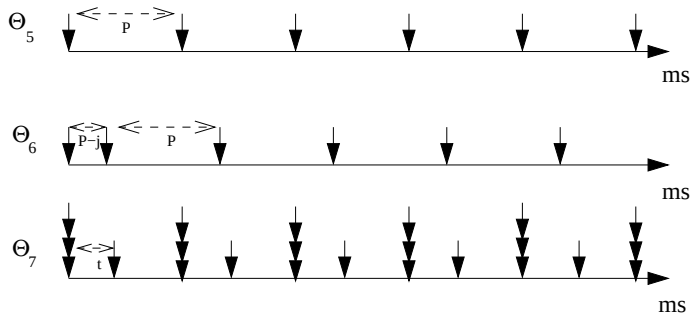

FIGURE 2.3.3. Example event streams ([60])

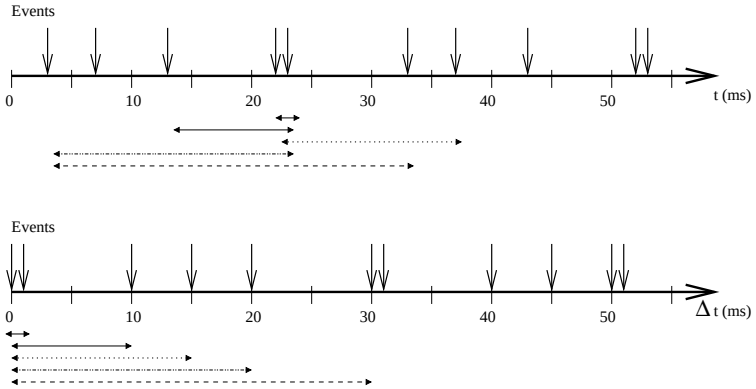

FIGURE 2.3.4. Transformation periodic event sequence into event stream

Event streams can describe all these examples in an easy and intuitive way. The offset value of the first event element is always zero. The reason is that this value models the shortest interval in which one single event can occur. As the position of the interval can be chosen freely it is possible to choose the start of the interval just before the event and the end just after the event. Then the length of the interval can be an infinitely small value being notated as zero for the purpose of simplicity.

Each homogeneous periodic event sequence can be transferred to an event stream with an equal number of elements by reordering the event elements (and recalculating the offsets). Note, that for the purpose of real-time analysis it is not necessary to extract the event stream out of a periodic event sequence, because the event stream can be directly extracted out of the system description in most cases. Let us, for example, consider again $\Theta_2 = \{(6ms, 2ms), (6ms, 3ms), (6ms, 5ms)\}$. The corresponding event stream is $\Theta_2 = \{(6ms, 0ms), (6ms, 1ms), (6ms, 3ms)\}$. The transfer from a homogeneous periodic event sequence to the corresponding event stream keeps the number of event elements and the period the same. For $\Theta'_4 = \{(30ms, 3ms), (30ms, 7ms), (30ms, 13ms), (30ms, 22ms), (30ms, 23ms)\}$ the corresponding event stream reads as $\Theta'_{4a} = \{(30ms, 0ms), (30ms, 1ms), (30ms, 10ms), (30ms, 15ms), (30ms, 20ms)\}$. The transformation of the periodic event sequence $\hat{\Theta}'_4$ to the event stream $\hat{\Theta}'_{4a}$ is visualized in figure 2.3.4. One event can occur within an interval of an infinitely small length so the first event element has an offset of

zero. Two events can occur in the original sequence at shortest between the time points 22 ms and 23 ms. The offset for the second event element is therefore one. All other distances between two events as for example between three and seven or between 23 ms and 33 ms (the second event of the first event element) are longer. Three events can occur at shortest between the time points three and 13 ms or between the time points 13 ms and 23 ms. In both cases the interval length, in which three events can occur, is 10 ms and therefore the offset of the third event element of Θ'_{4a} is 10 ms. The fourth event element has the offset 15 ms, that is the interval between the time points 22 ms and 37 ms. In this case the smallest interval containing four events crosses the boarder of the period. The distance between 3 ms and 22 ms would be 19 ms, between 7 ms and 23 ms would be 16, between 13 ms and 33 ms would be 20 ms and between 23 ms and 43 ms would also be 20 ms. All these interval lengths are longer than 15 ms and are therefore not the worst case.

With an infinite number of elements it would be possible to describe each set of events by an event stream. With a bounded number of elements it is necessary that the event sequence is either bounded too or has a part that is repeated periodically. For some sets of events the number of elements required to describe the exact occurrence of the events can become quite large. For example a burst of 100 events that is repeated periodically would require 100 event elements for description. To solve this problem we will extend the event streams to hierarchical event streams (or event spectra) in chapter 7.

For the purpose of evaluation it is not necessary to find the exact minimum intervals. It is sufficient to find for all intervals a lower bound. Additionally it is possible to simplify an event stream by using a pessimistic description with fewer elements than in the original event stream.

LEMMA 2.3.8. *Let Γ be a task set having tasks τ each one activated by a periodic event sequence Θ. The utilization of this task set is given by:*

$$U_\Gamma = \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau} \frac{c_\tau^+}{p_\theta}$$

For task sets having only homogeneous event streams without infinite periods activating the tasks a more simpler calculation is possible. Let p_Θ be the common period of all event elements of Θ. The utilization is then given by:

$$U_\Gamma = \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\Theta} |\Theta|$$

Let us, for example, consider a task set Γ with one task τ activated by the periodic event sequence Θ_4 of example 2.3.6. Let us assume a worst-case execution time $c_\tau^+ = 4$ ms. The utilization U_Γ can be calculated by $U_\Gamma = \frac{4}{10} + \frac{4}{15} = 66.6\%$. Using its homogeneous counter-part Θ_{4a} leads to the following calculation: $U_\Gamma = \frac{4}{30} \cdot 5 = 66.6\%$. The calculation for the event stream and for its homogeneous counter-part leads to the same results for the utilization. The utilization is independent of the offset and therefore the utilization of a periodic event sequence is the same as the utilization of the corresponding event stream.

LEMMA 2.3.9. *(Demand Bound Function - event stream version) Let Γ be a task set having tasks τ each one activated by an event stream Θ_τ. The maximum demand of the*

task set for each interval Δt can be calculated by:

$$\delta(\Delta t,\Gamma) = \sum_{\forall \tau \in \Gamma} \eta(\Delta t - d_\tau, \Theta_\tau) c_\tau^+ = \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t \geq a_\theta + d_\tau}} \left\lfloor \frac{\Delta t - a_\theta - d_\tau}{p_\theta} + 1 \right\rfloor c_\tau^+$$

PROOF. Follows directly out of the definitions. □

The demand bound function can be tested for every interval whether it exceeds the intersection ($\delta(\Delta t,\Gamma) \leq \Delta t$). If this is not the case for every possible interval, the task set is feasible.

Same as for the periodic task set, only those intervals are of relevance for which the value of the demand bound function changes. This set of test intervals is given by the set of all intervals for all event stream elements with

$$\Delta t = a_\theta + k p_\theta + d_\tau$$

with $k \in \mathbb{N}_0$. Consider again the event stream $\Theta_{4a} = \{(30\,ms, 0\,ms),\ (30\,ms, 1\,ms),\ (30\,ms, 10\,ms),\ (30\,ms, 15\,ms),\ (30\,ms, 20\,ms)\}$ and assume a deadline $d_{\Theta'_4} = 7$ ms. The relevant intervals would be $7\,ms$; $8\,ms$; $17\,ms$; $22\,ms$; $27\,ms$; $37\,ms$; $38\,ms$; $47\,ms$; $52\,ms$; $57\,ms$; $67\,ms$; $68\,ms$; ... .

Gresser [60, 61] proposes a schedulability analysis using the demand bound function. In contrary to the processor demand analysis he did not use a maximum test interval. Instead he proposes an alternating test of the demand bound function and a special upper bound of the demand bound function. This leads to unnecessary effort compared to the processor demand test.

A better way would be to calculate a maximum test interval for the event stream model. It is possible to modify the existing test interval of Ripoll et al. [119] and, of cause, also the one of Baruah et al. [19], to work with the event stream model. Let us re-consider again the proof for the test interval of Ripoll et al:

$$\Delta t \leq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t \geq a_\theta + d_\Theta}} \left\lfloor \frac{\Delta t - a_\theta - d_\tau}{p_\theta} + 1 \right\rfloor c_\tau^+$$

$$\Delta t \leq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t \geq a_\theta + d_\Theta}} \left(\frac{\Delta t - a_\theta - d_\tau}{p_\theta} + 1 \right) c_\tau^+$$

$$\Delta t \left(1 - \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t \geq a_\theta + d_\Theta}} \frac{c_\tau^+}{p_\theta} \right) \leq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t \geq a_\theta + d_\Theta}} \left(\frac{-a_\theta - d_\tau}{p_\theta} + 1 \right) c_\tau^+$$

Limiting Δt on $\Delta t \geq \max(a_\theta + d_\theta)$ leads to the following maximum test interval:

$$\Delta t \quad \leq \quad \frac{\sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau} \left(1 - \frac{a_\theta + d_\tau}{p_\theta} \right) c_\tau^+}{1 - U_\Gamma}$$

For the example Θ_{4a} with $c^+_{\Theta'_4} = 4$ ms and $d_{\Theta'_4} = 7$ ms the maximum test interval has the value:

$$\Delta t_{max} = \frac{\left(1 - \frac{0ms+7ms}{30ms}\right) \cdot 4ms + \left(1 - \frac{1ms+7ms}{30ms}\right) \cdot 4ms + ... + \left(1 - \frac{20ms+7ms}{30ms}\right) \cdot 4ms}{1 - \frac{2ms}{3ms}} = 13.8ms$$

Remember that this maximum test interval is only valid, if it is larger than $max_{\substack{\forall\tau\in\Gamma \\ \forall\theta\in\Theta}}(a_\theta + d_\tau)$. This result completes the schedulability analysis of event stream based task sets using EDF scheduling. We only have to check whether $\delta(\Delta t, \Gamma) \leq 1$ for all intervals Δt up to the maximum of the maximum test interval and $max_{\substack{\forall\tau\in\Gamma \\ \forall\theta\in\Theta}}(a_\theta + d_\tau)$.

The complexity of the processor demand analysis for event streams is quite similar to the complexity for the periodic task model. The only difference is that the set of test intervals for each task can be larger.

The processor demand analysis for event streams where n is the number of tasks, e is the maximum number of event elements for any event stream activating one of the tasks and $max(p_\tau - d_\tau)$ the maximum difference between the period and the deadline for any of the tasks has a complexity of $O(n \cdot e \cdot max(p-d))$ if $U_\Gamma \leq c$ and c is a fixed constant $0 < c < 1$. Using the maximum test interval of Ripoll et al. [119] leads to a complexity in an equal range.

For the sake of completeness we will now discuss the inverse function for the event bound function; the interval bound function.

DEFINITION 2.3.10. *The interval bound function calculates for a given number of events the minimum interval in which this number can occur:*

$$\psi(n, \Theta) = min(\Delta t | (\eta(\Delta t, \Theta) = n))$$

This function is required in the following chapters. For a homogeneous sequence evaluating the interval bound function is easy. It is only necessary to calculate first the number of totally completed periods. This can be done by dividing the given number of events by the number of those events generated by the sequence within a single period. The remaining events are then distributed on the event elements in the ascending order of their offsets. The interval is given by the sum of first the number of completed periods multiplied with the length of the period and of second the offset of the last event element generating an event.

Let Θ be a homogeneous event stream with $|\Theta_\infty|$ be the number of elements with an infinite period. Let Θ first contain the elements with the infinite period and after them the remaining elements with the common period p. Let the elements be sorted within each of both groups by their ascending offset a. Let Θ_k denote the $k-$th element of Θ. Let $mod(x,y) = x - \left\lfloor \frac{x}{y} \right\rfloor y$. We can rewrite $\psi(n, \Theta)$ by

$$\psi(n, \Theta) = \begin{cases} a_{\Theta_n} & n \leq |\Theta| \\ \left\lfloor \frac{n-|\Theta_\infty|}{|\Theta|-|\Theta_\infty|} \right\rfloor p_\Theta + a_{\Theta_{(|\Theta_\infty| + mod(n-|\Theta_\infty|, |\Theta|-|\Theta_\infty|))}} & else \end{cases}$$

In general it is necessary to use an optimization approach to find the correct distribution of the events on the different event elements.

Mathematically the concept behind the events stream model can be defined by the concept of sub-additive functions.

DEFINITION 2.3.11. *[31] (Sub-additive function) A function f is sub-additive if f is a monotonic increasing function and if and only if $f(I+J) \leq f(I)+f(J)$ for all $I,J \geq 0$*

In other words:

LEMMA 2.3.12. *The event bound function of an event stream is a sub-additive function.*

PROOF. Assume the above condition does not apply. In this case there would exist an interval $\Delta t + J$ having more events than Δt and J together. By definition both intervals contain each the maximum number of those events that can be contained by any interval of their respective length. In case that the complete interval $\Delta t + J$ is split into two intervals Δt and J, all events of $\Delta t + J$ have to exist in one of the parts. The assumption would therefore require that at least one part $\Delta t, J$ contains less events than its respective counterpart within the sum. This is contradiction to the definition of the event stream and therefore not possible. □

The event stream model is a flexible model to describe the stimuli of an embedded real-time system. The event stream describes an upper bound on the density of events. It is allowed for events to occur later and with less density than predicted by the event stream.

This lateness not only has an impact on the density of events before the arrival of the late event but of course also on the density of events behind this arrival. It is not allowed that such lateness leads locally to a higher density of events than allowed by the event stream. The event stream was constructed in a way that it includes for each interval an upper bound of the maximum number of those events that can occur within this interval. This is independent of the concrete start and end points of the intervals and also of the concrete scenario generating the events. Generally spoken, it is necessary to consider the densities for each possible scenario. For each scenario the maximum possible number of events can be calculated for each interval separately. The event stream function is an upper bound for the maximum number of events occurring within the interval in any possible scenario. The worst case of adjacent intervals can result of totally different scenarios.

Therefore in those cases where the lateness of an event would lead to a higher density in the part following the event, some of the following events have also to be delayed to keep the densities on the allowed level.

The event stream model is a general model but the efficient description of event stimuli containing bursts of events can require many elements.

2.3.3. Sporadically periodic tasks. Bursts consist of a number of events arriving within a short amount of time. They are normally followed by a time interval with no events or a much lower density of events. To extend the periodic task model with the capabilities to model bursts the sporadically periodic task model was introduced by Audsley et al. [8, 126]. Two different periods are assigned to one task, an inner period to model the distances between the events within a burst and an outer period to model the distances

between the bursts. Additionally the length of the burst is limited by a maximum number of events.

As the periodic task model this model is also limited in its capabilities too. Only simple periodic bursts can be described. The hierarchical event stream model (chapter 7) will extend the event stream model with the capabilities to model all kind of bursts. It is therefore a generalization of this model and of the event stream model.

2.3.4. Periodic with jitter and minimum separation. Richter et al. [117] proposed an extension of the periodic task model by adding a minimum separation distance between two consecutive events. A task τ is given in this model by $\tau = (p, j, s, c^+, d)$ where p is the period, j is the jitter, s is the minimum separation time, c^+ is the worst-case execution time and d is the relative deadline of the task.

The reason for this extension is the holistic schedulability analysis of distributed systems proposed by Tindell & Clark [130] and improved by Redell et al. [113, 114, 115]. This model consists of chains of tasks allocated on a set of resources with a fixed distribution of the tasks on the resources. The initial tasks of the chains are triggered by periodic stimulis with jitter and each task generates an event at the end of each of its jobs. These outgoing events activate the following tasks in the chain that can be assigned to a different resource. The activating event sequences for the following tasks are also modeled by a period and a jitter. In the holistic approach the period of the incoming event sequence for a following task is the same as the period of the previous task. The jitter instead is increased by the difference of the best-case and the worst-case response time of the previous task. A job can finish somewhere between the best-case response time and the worst-case response time of its task, so the jitter of the following task has to cover the space between these response times. The response time of the following task depends directly on its jitter value as proved in section 2.1. A larger jitter leads to a larger response time. The last task of the chain can have large jitter values, so the analysis can lead to task sets in which the jitter of a task can be several times larger than the period of this task.

The problem of the simple periodic model is that it assumes that all events of the same task within the jitter interval can occur simultaneously. It is obvious that this simultaneous occurrence of events cannot happen in the scenario of above as the jobs of one task are processed one after the other and therefore two outgoing events of the same task are separated at least by the best-case execution time of the task. This was the reason for introducing the minimum separation distance in the model proposed by Richter et al. [117].

With the new model the minimum separation distance can separate the jobs of the following task, so that they can be handled one after the other. This reduces the required resources for the following tasks. Despite that this model is a clear advantage to the periodic model it does not have the general modeling capacities of the event stream model. Unfortunately the separation condition cannot be modeled efficiently with the event stream model. This problem will also be solved with the hierarchical extension of the event stream model (chapter 7).

2.3.5. Streaming application model. A more expressive model for bursts was proposed by Chakraborty and Thiele in [40]. An event stream is described by a set of elements each containing an interval and the maximum number of events which can occur within this interval. The different elements limit each other, so that only the minimum number of events allowed by all elements can occur in the event stream. The main problem of this model is the expensive evaluation, as it is necessary to build for evaluation all possible linear combinations of each possible pair of elements. It can also not model efficiently event streams with several bursts having different minimum separation times within the bursts.

A related but less expressive approach is the rate-base execution model [56, 57, 67]. In it each task has as parameters only one interval and the number of events generated at most within this interval.

2.3.6. Real-Time calculus. A methodology for the real-time analysis of a network of modules is the real-time calculus by [39, 127, 128, 131]. It is based on the network calculus approach defined by [43] and [101]. The real-time calculus is an approach for a compositional real-time analysis based on the concept of the min-plus and the max-plus algebra. It splits the whole distributed system into processing components having an incoming upper and lower arrival and upper and lower capacity curve and provides the equations to calculate the outgoing upper and lower arrival curves and the remaining upper and lower capacity curve out of theses incoming curves.

The event pattern is modeled by an arrival curve $\alpha_f(\Delta t)$ denoting the number of those events arriving within a time interval of length Δt. For this function $\alpha_f^u(\Delta t)$ denotes the upper bound and $\alpha_f^l(\Delta t)$ the lower bound for the arrival curve. These functions deliver for every Δt the maximum respective the minimum number of those events which can occur in any interval of length Δt. Therefore these functions are also sub-additive functions as the event bound function of the event stream model.

The real-time calculus also defines service curves $\beta_r(\Delta t)$ similar to the arrival curves. They model the number of computational requirements that can be handled by the resource during a time interval of length Δt. Again $\beta_r^u(\Delta t)$ and $\beta_r^l(\Delta t)$ define the upper and lower bound of the service curve.

The processing ability is measured in computation time of an idealized resource. This is in consistency to previous work in real-time analysis where also computation time is used to measure the computational effort required by a resource.

To explain the functionality of the real-time calculus approach figure 2.3.5 shows an example scheduling network. A scheduling network is a system consisting of several chains of tasks and a set of resources. Each task τ of the task chain is mapped to one resource ρ. The tasks mapped on the same resource are scheduled with fixed priority scheduling. Different tasks of a chain can be mapped on different resources. In the figure 2.3.5 the tasks τ_1, τ_4, τ_6 form a task chain and the tasks τ_1, τ_4, τ_7 form another task chain. Each task τ is activated by an upper and lower arrival curve $\alpha_\tau^u(\Delta t)$ and $\alpha_\tau^l(\Delta t)$ and the available computational effort for this task is described by an upper and lower service curve $\beta_\tau^u(\Delta t)$ and $\beta_\tau^l(\Delta t)$.

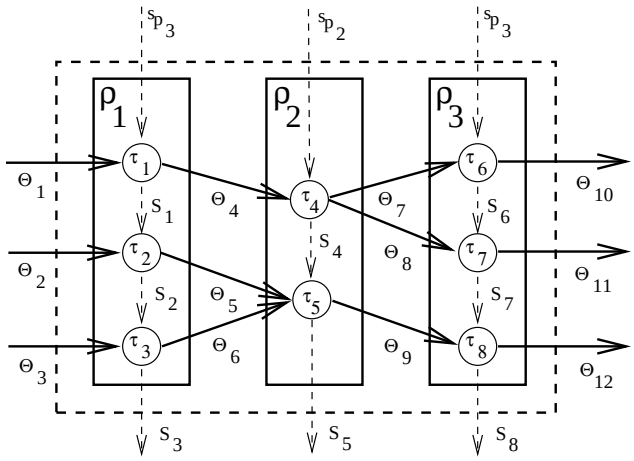

FIGURE 2.3.5. Scheduling network for real-time calculus

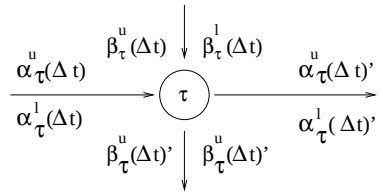

FIGURE 2.3.6. Real-Time Calculus of single task

Each task of the system is considered as a so-called greedy processing component (GPC). Figure 2.3.6 gives a closer look on a greedy processing component (GPC). For each task we have an incoming (upper and lower) arrival curve $\alpha^u_\tau(\Delta t)$ and $\alpha^l_\tau(\Delta t)$ modeling the workload for τ. We also have an (upper and lower) service curve $\beta^u_\tau(\Delta t)$ and $\beta^l_\tau(\Delta t)$ modeling the amount of workload that can be handled by the resource.

The analysis of a task generates outgoing (upper and lower) arrival ($\alpha^u_\tau(\Delta t)'$ and $\alpha^l_\tau(\Delta t)'$) and service curves ($\beta^u_\tau(\Delta t)'$ and $\beta^l_\tau(\Delta t)'$). The outgoing arrival curve is a modification of the incoming arrival curves and is also the incoming arrival curve of the following task in the chain. The outgoing service curve is the incoming service curve reduced by the workload handled by the task. It is the incoming service curve for the task with the next lower priority on the same resource.

The real-time calculus provides the equations to describe the relationships between the incoming and outgoing curves [39]. The curves can be of two types, event based or resource based. The arrival curves are normally event based and the service curves are resource based. Each of the two types can be transferred into the other. The curves used for each equation have to be of the same type. As the arrival curves are event based and the service curves are resource based it is preferable to use the event-based types for calculating the outgoing arrival curves and the resource-based type for the outgoing service curves. For the calculation the functions sup and inf are needed. The functions give the maximum or minimum value. The difference to the function *max* and *min* is that sup and inf provides upper and lower bounds. Their value can be reachable, but need not to be.

The outgoing upper arrival curve of the basic module in the real-time calculus, the greedy processing component (GPC), is given by [39]:

$$\alpha_\tau^u(\Delta t)' = \min(\inf_{0\leq\Delta t'\leq\Delta t}\{\sup_{0\leq v<\infty}[\alpha_\tau^u(\Delta t'+v)-\beta_\tau^l(v)]+\beta_\tau^u(\Delta t-\Delta t')\},\beta_\tau^u(\Delta t))$$

The outgoing arrival of τ is limited by the available capacity $\beta_\tau^u(\Delta t)$. No more workload can be handled within Δt despite how much has arrived. In case the maximum available capacity is not reached we have to split the total interval Δt into two parts $\Delta t'$ and $\Delta t-\Delta t'$. $\Delta t'$ is the part in which the system is not completely utilized, $\Delta t-\Delta t'$ is the part in which a full utilization occurs. The splitting point is the last idle point of the system, e.g. the last point in which no processable workload is available. Therefore for the second part we can limit the outgoing arrival flow by the available capacity $\beta_\tau^u(\Delta t-\Delta t')$. For the first part all available workload is handled.

The available workload consists of the workload arriving within $\Delta t'$ and the maximum available workload at the start of Δt. This is that amount of workload that has arrived previously but was not processed by the system so far. The calculation of these workloads is done together by $\sup_{0\leq v\leq\infty}[\alpha_\tau^u(\Delta t'+v)-\beta_\tau^l(v)]$. It is necessary to find the interval v providing the maximum amount of remaining workload. This is workload that can arrive within v at most ($\alpha_\tau^u(v)$) reduced by the workload being processed within v at least ($\beta_\tau^l(v)$). The intervals v and $\Delta t'$ are considered together. Therefore the workload available to handle within $\Delta t'$ is calculated by the maximum amount of workload arriving within $\Delta t'+v$ reduced by the minimum amount of workload processed within v. To find the maximum amount of workload for $\Delta t'$ we have to consider every possible interval v to get an upper bound for $\alpha_\tau^u(\Delta t'+v)-\beta_\tau^l(v)$.

The interesting splitting interval $\Delta t'$ of Δt is given by the minimum of the calculated workloads for all $\Delta t'$. It is the last idle point of the system. The reason why we have to use it is that all other splitting points overestimate the processable workload as we assume a total utilization for the remaining interval $\Delta t-\Delta t'$. Note that this value is bounded by the maximum available capacity within Δt ($\beta_\tau^u(\Delta t)$).

The outgoing lower arrival curve is given by [39]:

$$\alpha_\tau^l(\Delta t)' = \inf_{0\leq\Delta t'\leq\Delta t}\{\alpha_\tau^l(\Delta t')+\beta_\tau^l(\Delta t-\Delta t')\}$$

For the outgoing lower arrival curve the interval Δt is again split into two parts $\Delta t'$ and $\Delta t-\Delta t'$. $\Delta t'$ is again the part that is not fully utilized (with regard to the lower incoming service curve $\beta_\tau^l(\Delta t)$). $\Delta t-\Delta t'$ is the fully utilized part (again with regard to the lower incoming service curve $\beta_\tau^l(\Delta t)$). The minimum outgoing arrival is given by the minimum sum of these parts for all intervals $\Delta t'$.

The outgoing service curves, which give the available capacity for the task with the next lower priority on the same processor, can be calculated by [39]:

$$\beta_\tau^l(\Delta t)' = \max(\sup_{0\leq\Delta t'\leq\Delta t}\{\beta_\tau^l(\Delta t')-\alpha_\tau^u(\Delta t')\},0)$$

$$\beta_\tau^u(\Delta t)' = \max(\sup_{0\leq\Delta t'\leq\Delta t}\{\beta_\tau^u(\Delta t')-\alpha_\tau^l(\Delta t')\},0)$$

The minimum remaining capacity $\beta_\tau^l(\Delta t)$ can be found in an interval $\Delta t'$ for which the minimum available capacity exceeds the maximum arriving workload. Only if this is the case some capacity is remaining. The maximum calculated value for all intervals $\Delta t'$ is the overall remaining capacity. For the interval $\Delta t - \Delta t'$ the maximum arriving workload requires fully the minimum available capacity, therefore this interval does not contribute to the remaining capacity. Note, that capacity remaining for some intervals $\Delta t'$ is also remaining for any larger intervals because none of the arriving workload can be executed in the past. So, even if the total workload for Δt exceeds the available capacity within Δt there can still exist remaining capacity for Δt.

For the maximum remaining capacity we have the same scheme but we are using the maximum available capacity and the minimum required workload.

For example in [131] the convolution and the deconvolution are defined as central operations in the theory of the min-plus and max-plus algebra. Let γ be either α or β.

DEFINITION 2.3.13. Min-plus convolution $\otimes$ / deconvolution $\oslash$

The min-plus convolution $\gamma_C = \gamma_A \otimes \gamma_B$ and the min-plus deconvolution $\gamma_C = \gamma_A \oslash \gamma_B$ is given by:

$$\gamma_A(\Delta t) \otimes \gamma_B(\Delta t) = \inf_{0 \le \Delta t' \le \Delta t} \{\gamma_A(\Delta t - \Delta t') + \gamma_B(\Delta t')\}$$
$$\gamma_A(\Delta t) \oslash \gamma_B(\Delta t) = \sup_{0 \le \Delta t' < \infty} \{\gamma_A(\Delta t + \Delta t') - \gamma_B(\Delta t')\}$$

DEFINITION 2.3.14. Max-plus convolution $\overline{\otimes}$ / deconvolution $\overline{\oslash}$

The max-plus convolution $\gamma_C = \gamma_A \overline{\otimes} \gamma_B$ and the max-plus deconvolution $\gamma_C = \gamma_A \overline{\oslash} \gamma_B$ is given by:

$$\gamma_A(\Delta t) \overline{\otimes} \gamma_B(\Delta t) = \sup_{0 \le \Delta t' \le \Delta t} \{\gamma_A(\Delta t - \Delta t') + \gamma_B(\Delta t')\}$$
$$\gamma_A(\Delta t) \overline{\oslash} \gamma_B(\Delta t) = \inf_{0 \le \Delta t' < \infty} \{\gamma_A(\Delta t + \Delta t') - \gamma_B(\Delta t')\}$$

The above curves can also be calculated using the convolution and deconvolution:

$$\alpha_\tau^{u'} = \min\{(\alpha_\tau^u \otimes \beta_\tau^u) \oslash \beta_\tau^l, \beta_\tau^u\}$$
$$\alpha_\tau^{l'} = \min\{(\alpha_\tau^l \oslash \beta_\tau^u) \otimes \beta_\tau^l, \beta_\tau^l\}$$
$$\beta_\tau^{l'} = \max\{(0 \otimes (\beta_\tau^l - \alpha_\tau^u), 0)\}$$
$$\beta_\tau^{u'} = \max\{(0 \otimes (\beta_\tau^u - \alpha_\tau^l), 0)\}$$

The calculation for the outgoing curves is based only on the incoming curves and the values of the task itself. No other values are required. This makes it possible to consider each task separately from each other. Each incoming arrival or service curve of the scheduling network is an incoming arrival or service curve of one tasks of the scheduling network. These curves are transformed with the transformation equations of these tasks to new arrival and service curves. These can be incoming curves of other tasks of the scheduling network, transformed again for the next following tasks and so on.

It is possible to calculate a value somewhere in the scheduling network by recursively evaluating the transformation equations until the original incoming curves are reached. Obviously such an approach is expensive to compute. So, the equations describing the relationship between the functions are expensive to compute for general functions, too. To calculate the modification equations independently from each other, an event model is needed that can characterize the modified curves.

No concrete description for the functions themselves is provided in the real-time calculus. As the time t is defined for all values up to ∞ it is not possible to simply enumerate all values. A good finite description for this function is necessary. The complexity of the relationship equations depends directly on the complexity of this description.

In [39, 76] an approximation for the arrival and service curves was proposed in which each curve is described by three straight line segments. One segment describes the initial offset or arrival time, one a possible initial bursts and one the long time rate. As outlined in [6] this approach is to inaccurate to be suitable for complex systems.

In [131] an exact characterization for the curves is given. The curves are described by two sets of segments, one set for the aperiodic first part of the curve and one describing the periodic part. The number of segments in each of the parts is not bounded in general. Operations performed on two curves with different periods and/or different length of the aperiodic parts requires the hyper-period as the period of the resulting stream. In the worst case, the hyper-period is the product of all periods. This can lead to an explosion of event segments and an uncontrollable run-time of the proposed algorithms.

The new model described in this work is quite suitable to be an efficient description for the real-time calculus. Therefore the real-time calculus is extended to an event stream calculus in this work.

We will discuss this model later and show how an new extended version of the event stream model, the hierarchical event streams can be used to achieve an explicit description for the curves of the real-time calculus.

CHAPTER 3

Approximation for dynamic priorities

To overcome the problem of the evaluation complexity as described in section 2.2.4 we will propose in the following an approximative schedulability test. Approximative tests are inexact tests. A range of systems exists for which the tests may not be able to answer the question whether the systems meets all deadlines or not. The advantage to use such tests is that the approximative tests run much faster than their exact counterparts and therefore allow to solving problems that would be unsolvable otherwise because of their complexity.

The main characteristic of an approximative algorithm is, that the area of uncertainty is bounded. Only for the exact solution and solutions close to the exact one the algorithm might not be able to make a decision.

Take for example the knapsack problem. Given is a problem instance

$$\{b, w_1, w_2, ..., w_n, c_1, c_2, ..., c_n\}$$

There exists a set of goods having different sizes w_i and costs c_i. The problem is to pack a knapsack having a size b with these goods getting as much value into the knapsack as possible. In its exact general form this problem is a NP-hard optimization problem [66].

But nevertheless the knapsack problem becomes tractable with a small modification of the requirements [66]. We do not consider a knapsack of size b but allow that the size of the knapsack can vary with an error ε. The knapsack can have a size up to $b + b \cdot \varepsilon$. In this case an approximation exists that is solvable in polynomial time [66]. The resulting solution may not be optimal to a knapsack of the size b but it is optimal to at least one knapsack having a size within the interval $[b, b + b \cdot \varepsilon]$. So even if we have not found the optimal solution for the knapsack in question we have found an optimal solution for a knapsack with a size close to the size of the knapsack in question. The result is therefore close to the optimal solution of the problem and can be reasonably used for practical applications.

Different types of approximations are available. There exist algorithms having a fixed maximum error depending on the problem. Better are those algorithms for which the error value and therefore the size of the area of uncertainty are freely selectable. It should be possible that the error can become as small as desirable. Of cause, a reduction of the error increases the worst-case run-time of the test in many cases.

Algorithms having a polynomial run-time complexity with respect to the problem size are called polynomial time approximation schemes. Algorithms having a polynomial run-time complexity with respect to both, the problem size and the value of the error are called fully polynomial-time approximation schemes [66]. They are normally the best kind of

approximation algorithm possible. Polynomial with respect to the error means that a decrease of the error by a certain factor k (for example reducing it by half) can only lead to an increase of the run-time that is polynomial bounded with respect to k.

The approximative analysis is only sufficient, but the degree of exactness is adjustable. An approximative analysis can have a much smaller complexity than the exact analysis with only a small change in the requirements. An optimization problem is considered tractable if there "exists a polynomial-time approximation that solves the problem with a reasonable error"[66].

A feasibility analysis is sufficient if it classifies all systems that do not fulfill the real-time requirement correctly. An infeasible schedule is always recognized as infeasible. It is only possible that a sufficient test does not recognize a schedulable system as schedulable; it will never classify a non-schedulable system as schedulable.

In [37, 38] it was proposed to consider two other kinds of approximations, those being only necessary and those being neither necessary nor sufficient. Necessary tests give a correct classification for non-schedulable systems, whereas tests being neither necessary nor sufficient cannot guarantee either side. The idea behind considering these two additional types of approximations is that due to the limitation of the approximation error the uncertainty only affects a small number of systems being located close to the boarder between schedulable and non-schedulable systems. The assumption is that non-schedulable systems close to this boarder do only lead to small misses of deadlines. We will consider in the following only sufficient approximations. The proposed approximations therefore always guarantee the real-time requirement despite of their uncertainty.

In the following we will present an approximative schedulability analysis for single processor systems scheduled with EDF scheduling. It is the first approximation for this problem fulfilling the definition of a fully polynomial-time approximation scheme.

The main idea is to limit the maximum number of test intervals for each task separately by constructing an approximated demand bound function for each task and to add all approximative functions resulting in an approximated demand bound function for the task set.

The idea behind the approximated demand bound function for a task is to analyze for each function only a limited number of jobs (the first k jobs) exactly. The following jobs are approximated. We call the interval that includes all non-approximated jobs (e.g. the boarder between the non-approximated and the approximated jobs) the maximum exact test interval. For the approximation we use the specific utilization of the task. In the following we will give the formal definition of the approximated demand bound function.

3.1. Periodic task system

Let us first consider the approximation for the periodic task system as given in section 2.3.1. We have a task set Γ and tasks $\tau \in \Gamma$ with $\tau = (p, a, c^+, d)$ where p is the period, a is the offset, c^+ is the worst-case execution time and d is the relative deadline of τ. The demand bound function $\delta(\Delta t, \Gamma)$ gives the maximum demand on computation time for a task set Γ within any interval of length Δt and the demand bound function $\delta(\Delta t, \tau)$ gives the part

of the maximum demand which is generated by task τ. Therefore the sum of the demand bound functions of all tasks of a task set gives the complete demand bound function of the task set ($\delta(\Delta t, \Gamma) = \sum_{\forall \tau \in \Gamma} \delta(\Delta t, \tau)$). The effort to calculate these demand bound functions depends not only on the number of tasks but also on the concrete parameters of the tasks. These are the period, deadline and worst-case execution time. Especially the ratio between the values of the largest and the smallest task in the task set contributes substantially to the computational effort required for analyzing the task set.

Consider for example the tasks $\tau_1 = (10\,s, 0\,ms, 5\,s, 7\,s)$ and $\tau_2 = (10\,ms, 0\,ms, 2\,ms, 3\,ms)$. Let us assume that we need a test bound Δt_{max} having a size of about ten times p_{τ_1}. Therefore $\Delta t_{max} = 100\,s$. We need ten test intervals for τ_1 ($7\,s, 17\,s, 27\,s, ..., 87\,s, 97\,s$). For τ_2 many more test intervals are necessary. The demand bound function has to be tested for the intervals: $3\,ms, 13\,ms, 33\,ms, ..., 99983\,ms, 99993\,ms$. In total we require $\frac{\Delta t_{max}}{p_{\tau_2}} = 10000$ test intervals for τ_2 compared to 10 for τ_1. Therefore τ_2 dominates the total number of test intervals. If we increase all parameters of τ_2 by the factor 10 the total number of test intervals will increase by the same factor. Reducing the parameters of τ_1 by the same factor will result in a reduction of the maximum test bound by nearly the same factor and therefore to a reduction of the number of test intervals required for τ_2 and of the necessary total number of test intervals. Therefore the total number of test intervals depends on the concrete values of the tasks and the fraction between the values of the largest and the smallest tasks.

To avoid these problems we propose to approximate the demand bound functions. The idea is to evaluate the first *k* test intervals (jobs) for each task exactly and approximate the remaining test intervals using the specific utilization of the task. In the following we first present the concepts and explain them informally. The formal proofs are given later in section 3.4 to cover a more general model.

DEFINITION 3.1.1. *Approximated demand bound function of a task*

Let τ be a task and k be a chosen number of those jobs that should be considered for the task exactly. Let $\Delta t_{\tau,k} = d_{\tau,k} = (k-1)p_\tau + d_\tau$. We call $\delta'(\Delta t, \tau, k)$ with

$$\delta'(\Delta t, \tau, k) = \begin{cases} \delta(\Delta t_{\tau,k}, \tau) + \frac{c_\tau^+}{p_\tau}(\Delta t - \Delta t_{\tau,k}) & \Delta t > \Delta t_{\tau,k} \\ \delta(\Delta t, \tau) & \Delta t \leq \Delta t_{\tau,k} \end{cases}$$

the approximated demand bound function. This formula can also be transformed in

$$\delta'(\Delta t, \tau, k) = \begin{cases} \frac{c_\tau^+}{p_\tau}(\Delta t - d_\tau) + c_\tau^+ & \Delta t \geq \Delta t_{\tau,k} \\ \delta(\Delta t, \tau) & \Delta t < \Delta t_{\tau,k} \end{cases}$$

Figure 3.1.1 shows an example of an approximated demand bound function $\delta'(\Delta t, \tau, k)$ with $k = 4$ and the comparable exact demand bound function $\delta(\Delta t, \tau)$. The first four test intervals are evaluated exactly and the remaining test intervals are approximated using a straight line. The slope of the line is the specific utilization of the task.

The interesting point of this approximation is the resulting approximation error. As shown in Figure 3.1.1 the maximum distance between the approximated and the real demand bound function is one time the worst-case execution time c_τ^+. The approximation

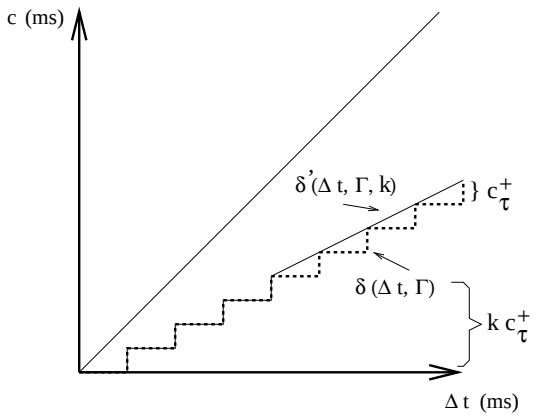

FIGURE 3.1.1. Approximation of a single task

starts at the k-th test interval. The demand bound function has for the k-th test interval a value of $k \cdot c_\tau^+$ therefore the demand bound function has for all those interval sizes for which an approximation error can occur at least this value. The relative approximation error is the maximum distance between the approximated and the exact demand bound function divided by the value of the exact demand bound function. The maximum approximation error is the maximum relative error. For the proposed approximation it is easy to find an upper bound to the approximation error:

$$\varepsilon_{\tau,k} = \frac{\delta'(\Delta t, \tau, k) - \delta(\Delta t, \tau)}{\delta(\Delta t, \tau)} \leq \frac{c_\tau^+}{k c_\tau^+} = \frac{1}{k}$$

The overall approximation error can be bounded by the same value:

$$\varepsilon = \frac{\delta'(\Delta t, \Gamma, k) - \delta(\Delta t, \Gamma)}{\delta(\Delta t, \Gamma)} \leq \frac{1}{k}$$

The proof follows in section 3.4. The error is independent of the parameters of the task and the system and does only depend on the selectable number of exactly analyzed test intervals. The value of the approximated function is for each interval Δt equal or larger than the value of the exact function for the same interval Δt. The analysis compares this value of the function with the available capacity. To proof schedulability the value of the demand bound function (approximated or not) has always to be equal or smaller than the available capacity.

The approximated demand bound functions of the single tasks can be added to an approximated demand bound function of the complete task set.

DEFINITION 3.1.2. *The approximated demand bound function of a task set* Γ *is the sum of the separated approximated demand bound functions of all tasks* $\tau \in \Gamma$.

$$\delta'(\Delta t, \Gamma, k) = \sum_{\forall \tau \in \Gamma} \delta'(\Delta t, \tau, k)$$

This definition is outlined in figure 3.1.2. For usability of these functions as approximation it is necessary for them to keep two conditions. First we have to guarantee that

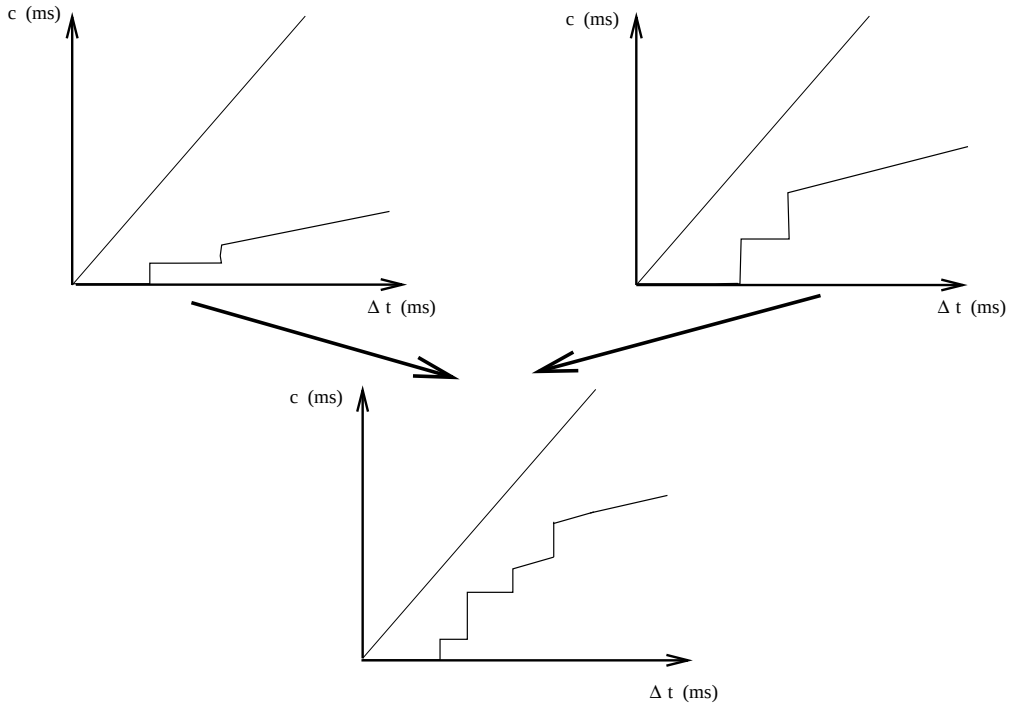

FIGURE 3.1.2. Adding two approximated demand bound functions

the demand is never underestimated by the approximation, which means the approximated demand bound function is always greater or equal the real demand bound function.

LEMMA 3.1.3. *Sufficiency condition*

The approximated demand bound function $\delta'(\Delta t, \Gamma, k)$ has for each possible interval Δt and each value of k at least the same amount of demand than the exact demand bound function $\delta(\Delta t, \Gamma)$.

$$\delta(\Delta t, \Gamma) \leq \delta'(\Delta t, \Gamma, k)$$

Second it is necessary that the error of the approximation is bounded.

THEOREM 3.1.4. *Let ρ_1 be a processor with a capacity given by the available execution time function $\chi(\Delta t, \rho_1)$ and ρ_2 be a processor with a slightly higher capacity given by $\chi(\Delta t, \rho_2) = \chi(\Delta t, \rho_1) + \frac{1}{k}\chi(\Delta t, \rho_1)$. If the schedulability analysis for ρ_1 using the exact demand bound function results "schedulable", the approximated demand bound function is guaranteed to result also "schedulable" if tested against the capacity function of processor ρ_2, the processor with the slightly higher capacity, so if*

$$\delta(\Delta t, \Gamma) \leq \chi(\Delta t, \rho_1)$$
$$\Rightarrow \delta'(\Delta t, \Gamma, k) \leq \chi(\Delta t, \rho_2)$$

The relative difference between the capacity of ρ_1 and ρ_2 is simply:

$$\frac{\chi(\Delta t, \rho_2) - \chi(\Delta t, \rho_1)}{\chi(\Delta t, \rho_1)} = \frac{\chi(\Delta t, \rho_1) + \frac{1}{k}\chi(\Delta t, \rho_1) - \chi(\Delta t, \rho_1)}{\chi(\Delta t, \rho_1)} \leq \frac{1}{k}$$

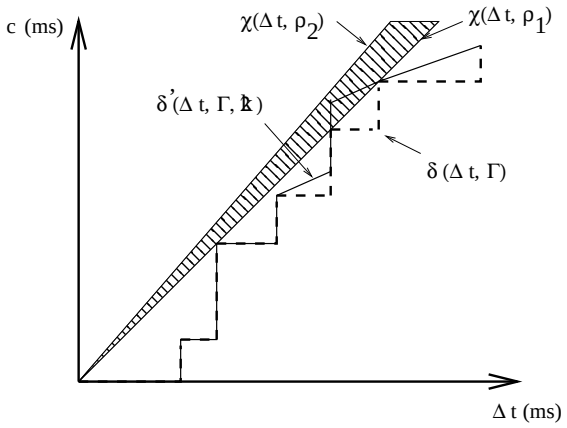

FIGURE 3.1.3. Visualization of the approximation bound

The proofs for the lemma and the theorem will follow also in section 3.4. The theorem 3.1.4 is visualized in figure 3.1.3. It shows an example of an exact and an approximated demand bound function. The approximation is done after the second test interval. The exact demand bound function $\delta(\Delta t, \tau)$ stays always below the capacity function $\chi(\Delta t, \rho_1)$ of ρ_1 while the approximated demand bound function $\delta(\Delta t, \tau, 2)$ exceeds this capacity function. But it does not exceed the area between this capacity function and the capacity function $\chi(\Delta t, \rho_2)$ of ρ_2. ρ_2 has only a slightly higher capacity than ρ_1. The difference between these capacities depends on the error value as defined in lemma 3.1.4.

For an efficient implementation of the test it is not necessary to calculate $\delta'(\Delta t, \Gamma, k)$ for each test interval Δt separately. The intervals can be calculated stepwise by only considering the differences between the single intervals. A possible implementation of the superposition analysis is shown in algorithm 1.

The algorithm works as follows: First it initializes the *test-list* with the first test interval of each task. The first test interval of a task is the smallest interval for which it is necessary to have completely processed one job of the task. The length of this interval is the smallest distance between the release time of any job of the task and the absolute deadline of this job and is exactly the relative deadline of the task. The worst-case scenario is that the invocation of the task and the start of the interval occur simultaneously. In this case the deadline of the task occurs simultaneously with the end of the interval and, of cause, processing the task has to be finished by this point of time. Therefore the interval with the length equal to the deadline of the task is the smallest interval in which the task has to be processed completely. This step has to be done for each task in the task set. The algorithm processes the list of test intervals in ascending order of their length. For each test interval it adds to the cumulated demand the computation time of the task belonging to the interval. Initially this demand is empty. If the cumulated demand exceeds the length of the actual interval, the analysis delivers *not schedulable.* In this case more computation time as being available within the interval is required to process all jobs within the interval. Therefore at least one job would miss its deadline.

Algorithm 1 Superposition Analysis

```
Algorithm Superposition
Given: task set Γ, k
IF U_Γ = Σ_{∀τ∈Γ} c_τ^+ / p_τ > 1
    ⇒ not schedulable
END IF
testlist := {}
Δt_old := 0
Δt_max := Σ_{∀τ∈Γ} (1 − d_τ/p_τ) c_τ^+ / (1 − U_Γ)
∀τ ∈ Γ: ADD te = (d_τ, τ) TO testlist
WHILE (Δt_old ≤ Δt_max)
    te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
    Δt =INTERVAL OF te
    τ =TASK BELONGING TO te
    REMOVE te FROM testlist
    δ' := δ' + c_τ^+ + U_r(Δt − Δt_old)
    IF (δ' > Δt)
        ⇒ not schedulable
    END IF

    IF (Δt = d_τ)
        ADD te = (Δt + p_τ − j_τ) TO testlist
    ELSE IF (Δt < (d_τ + p_τ k))
        ADD te = (Δt + p_τ, τ) TO testlist
    ELSE
        U_r := U_r + c_τ^+ / p_τ
    END IF
    Δt_old := Δt
END WHILE
⇒ schedulable
```

As next step it is checked whether the maximum number of exactly evaluated test intervals for the task is reached. If it is not reached, the next larger test interval of the task is added to *test-list*. It is calculated by adding the period of the task p_τ to the length of the actual test interval. Therefore *test-list* contains at most one test interval for each task at one time. If the maximum number of exactly evaluated test intervals for this task is reached, all further test intervals of the task are skipped due to the approximation. For compensation this skipping of test intervals an approximative value for the demand of the task has to be considered at each following test interval. Therefore when a task is approximated, its specific utilization $\frac{c_\tau}{p_\tau}$ is added to U_r. U_r contains the cumulated utilization of all approximated tasks and is initially empty. It is used to calculate the additional approximated execution times for all approximated tasks by

$$c^+_{add,\Gamma'} = U_r(\Delta t_{act} - \Delta t_{old})$$

The analysis ends and delivers *schedulable* if either *test-list* is empty (when all tasks are approximated) or if the maximum overall test interval Δt_{max} is reached.

The proof for the exactness of this algorithm and the questions of complexity and the error are discussed in the sections 3.4 and 3.6.

3.2. Capacity calculation for the period task model

During the system design process it is often necessary to choose a processor for a certain task set. Therefore it is interesting to calculate the capacity necessary for this task set. If we assume a scalable processing element for which the relationship between the execution times of all tasks is fixed, we can achieve such a calculation efficiently. We only need a small modification of the proposed superposition algorithm.

We need to assume a standard processing element to measure the execution time on it. The question for the calculation is, which ratio the capacity of the minimum required processing element has to the capacity of this standard processing element. We can still calculate the demand bound function in the same way using the standard processing element for measuring the execution times. But the calculation of each kind of the maximum test interval, with exception of the LCM (least common multiple) of the periods, requires the utilization of the task set and therefore the capacity of the resource. The way to avoid the maximum test interval is to use the approximation. We then need to calculate the demand bound function for each test interval required by the approximation. In case of an approximation error of 1% we have to calculate 100 test intervals for each task in the task set, in case of an error of 0.1% we need 1000 test intervals for each task and so on.

We then only have to calculate the utilization for each test interval separately and store the maximum one. It determines the necessary capacity for the processing element. The necessary capacity is at least the utilization of the task set. The algorithm 2 implements this idea.

C_{proc} is the capacity of the minimal processor necessary to handle the task set and fulfilling all real-time requirements. The main idea of the algorithm is to calculate for each test interval the dimension of the processor required to meet the real-time requirements of the test interval. We measure the dimension of the test intervals comparing its demand with a generic processing element. We call the demand of one test interval it's specific utilization and measure it in percent of the capacity of the generic processing element. The required dimension is the maximum of all specific utilizations. It can be quite larger than 100% leading to a larger processing element than the generic processing element.

We can improve the above algorithm by reconsidering the maximum test bound. Instead of using a fixed pre-calculated maximum test bound we will recalculate it every time the value of the necessary capacity changes. The calculation of the test interval differs from the known calculation, as the execution times are measured on a standard processor and not longer on the analyzed processor. Let us reconsider the maximum test interval given by Ripoll et al. [119] introduced in lemma 2.2.13 for the new defined execution times. Let C_N be the new capacity and $C_S = 1$ be the standard capacity in which the execution times are measured. Let U_Γ be the utilization of the task set compared to C_N. Then the maximum

Algorithm 2 Dimensioning a processing element

```
Algorithm DimProcSimple
Given: task set Γ, k
testlist := {}
C_result := U_Proc = Σ_{∀τ∈Γ} c_τ^+ / p_τ
Δt_old := 0;  δ' := 0
U_ready := 0
∀τ ∈ Γ: ADD te = (d_τ, τ) TO testlist
WHILE (test-list ≠ {})
    te=ELEMENT WITH SMALLEST Δt IN testlist
    Δt =INTERVAL OF te
    τ =TASK BELONGING TO te
    REMOVE te FROM testlist
    δ' := δ' + c_τ^+ + (Δt − Δt_old)U_ready
    C_act := δ'/Δt
    IF (C_act > C_result)
        C_result := C_act
    END IF

    IF (Δt < (d_τ + p_τ k))
        ADD te = (Δt + p_τ, τ) TO testlist
    ELSE
        U_ready := U_ready + c_τ^+ / p_τ
    END IF
    Δt_old := Δt
END WHILE
 ⇒ C_result
```

test interval looks as follows:

$$\Delta t_{max} := \frac{\sum_{\forall \tau \in \Gamma} \left(1 - \frac{d_\tau}{p_\tau}\right) \frac{C_S}{C_N} c_\tau^+}{1 - \frac{U_\Gamma}{C_N}} = \frac{\sum_{\forall \tau \in \Gamma} \left(1 - \frac{d_\tau}{p_\tau}\right) c_\tau^+}{C_N - U_\Gamma}$$

The only difference is the exchange of the capacity. In the original form we had a capacity of "1" now we have C_N. This is not a surprise. The maximum test interval depends obviously directly on the remaining capacity. If it gets very small the problem becomes more difficulty leading to a larger maximum test interval. Using this new maximum test interval we achieve the improved algorithm 3.

The hyper-period as maximum test bound, given in lemma 2.2.9 is independent of the capacity. The problem with this interval is that it becomes quite large in most cases and that we cannot limit the number of test intervals for this bound. We therefore do not consider this maximum test interval here.

3.3. Event Stream Model

In the following we will extend the superposition approximation to the advanced event stream model. Instead of the simple periodic activation of the tasks, the event streams allow a more powerful description of the stimuli of the tasks. Note, that the periodic task model is included as a special case in the event stream model. For the extension we have mainly

Algorithm 3 Dimensioning a processing element

```
Algorithm DimProcAdvanced
Given: task set Γ, k
testlist := {}
U_Γ := Σ_{∀τ∈Γ} c_τ^+ / p_τ
C_result := U_Γ
Δt_old := 0;  δ' := 0;
Δt_max := ∞;
U_ready := 0
∀τ ∈ Γ: ADD te = (d_τ, τ) TO testlist
WHILE (testlist ≠ {} AND Δt_old < Δt_max)
    te=ELEMENT WITH SMALLEST Δt IN testlist
    Δt =INTERVAL OF te
    τ =TASK BELONGING TO te
    REMOVE te FROM testlist
    δ' := δ' + c_τ^+ + (Δt − Δt_old) U_ready
    U_act := δ' / Δt
    IF (U_act > C_result)
        C_result := U_act
        Δt_max := max( (Σ_{∀τ∈Γ} (1 − d_τ/p_τ) c_τ^+) / (C_result − U_Γ), d_max )
    END IF

    IF (Δt < (d_τ + p_τ k))
        ADD te := (Δt + p_τ, τ) TO testlist
    ELSE
        U_ready := U_ready + c_τ^+ / p_τ
    END IF
    Δt_old := Δt
END WHILE
⇒ C_result
```

to exchange the demand bound functions for the periodic model by the corresponding functions for the event stream model.

Let us first repeat the demand bound function definition for the event streams:

$$\delta(\Delta t,\Gamma)=\sum_{\forall\tau\in\Gamma}\eta(\Delta t-d_\tau,\Theta_\tau)c_\tau^+=\sum_{\forall\tau\in\Gamma}\sum_{\substack{\forall\theta\in\Theta_\tau\\ \Delta t\geq a_\theta+d_\tau}}\left\lfloor\frac{\Delta t-a_\theta-d_\tau}{p_\theta}+1\right\rfloor c_\tau^+$$

Let θ be an event element belonging to the event stream Θ_τ which belongs to the task τ. For a single event element the demand bound function looks as follows:

$$\delta(\Delta t,\theta,\tau)=\begin{cases}\left\lfloor\frac{\Delta t-a_\theta-d_\tau}{p_\theta}+1\right\rfloor c_\tau^+ & : \quad \Delta t\geq a_\theta+d_\tau\\ 0 & : \quad else\end{cases}$$

So we have:

$$\delta(\Delta t,\Gamma)=\sum_{\forall\tau\in\Gamma}\sum_{\forall\theta\in\Theta_\tau}\delta(\Delta t,\theta,\tau)$$

Algorithm 4 Superposition Analysis for Event Streams

```
Algorithm SuperpositionEventStream
Given: task set Γ, k
IF U_Γ = Σ_{∀τ∈Γ} Σ_{∀θ∈Θ_τ} c_τ^+ / p_θ > 1
    ⇒ not schedulable
END IF
testlist := {}
Δt_old := 0
U_ready := 0
Δt_max := ( Σ_{∀τ∈Γ} Σ_{∀θ∈Θ_τ} (1 − (a_θ + d_τ)/p_θ) c_τ^+ ) / (1 − U_Γ)
∀τ ∈ Γ, ∀θ ∈ Θ_τ : ADD te = (NextInt(0, θ), θ) TO testlist
WHILE (testlist ≠ {} AND Δt ≤ Δt_max)
    te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
    Δt =INTERVAL OF te
    θ =EVENT-ELEMENT BELONGING TO te
    τ := τ_θ
    REMOVE te FROM testlist
    δ' := δ' + c_τ^+ + (Δt − Δt_old) U_ready
    IF (δ' > Δt_act)
        ⇒ not schedulable
    END IF

    IF (Δt < (d_τ + p_θ k))
        ADD Δt + p_θ, θ TO testlist
    ELSE
        U_ready := U_ready + c_τ^+ / p_θ
    END IF
    Δt_old := Δt
END WHILE
⇒ schedulable
```

We assume that exactly one event stream is connected to each task. Cases in which several independent event streams activate a task can be modeled either by unifying these event streams to one event stream or copying the task for each event stream.

For the calculation of the demand bound function the order of the event elements or even to which event stream a concrete event element belongs does not play a role. So we can simplify the above calculation by unifying the event stream elements of all tasks in one big set of event elements.

We can define an approximated demand bound function in the same way as we have done for the periodic task model.

DEFINITION 3.3.1. *Approximated demand bound function for an event element*

Let τ be a task activated by an event stream Θ having an event element θ and k be a chosen number of those test intervals that should be considered for the task exactly. Let $\Delta t_{\theta,k} = d_\tau + a_\theta + (k-1)p$. We call $\delta'(\Delta t, \theta, \tau, k)$ with

$$\delta'(\Delta t, \theta, \tau, k) = \begin{cases} \delta(\Delta t_{\theta,k}, \theta, \tau) + \frac{c_\tau}{p_\theta}(\Delta t - \Delta t_{\theta,k}) & \Delta t > \Delta t_{\theta,k} \\ \delta(\Delta t, \theta, \tau) & \Delta t \leq \Delta t_{\theta,k} \end{cases}$$

the approximated demand bound function for a single event element. Using $\delta(\Delta t_{\theta,k},\theta,\tau) = \frac{c_\tau^+}{p_\theta}(\Delta t_{\theta,k} - a_\theta - d_\tau)$ *we can also get:*

$$\delta'(\Delta t,\theta,\tau,k) = \begin{cases} \frac{c_\tau^+}{p_\theta}(\Delta t - a_\theta - d_\tau) & \Delta t > \Delta t_{\theta,k} \\ \delta(\Delta t,\theta,\tau) & \Delta t \leq \Delta t_{\theta,k} \end{cases}$$

The complete approximated demand bound function for the event stream model is the sum of the approximated demand bound functions for the event elements of the event stream:

$$\delta'(\Delta t,\Gamma,k) = \sum_{\forall\tau\in\Gamma}\sum_{\forall\theta\in\Theta_\tau} \delta'(\Delta t,\theta,\tau,k)$$

The error of each demand bound function for a single event element θ can be bounded separately to $\varepsilon_{\theta,k} = \frac{1}{k}$ leading to an overall error of also $\varepsilon_{\Gamma,k} = \frac{1}{k}$.

The approximated demand bound function for the event stream model has always the same or a larger value than the corresponding exact function. Therefore it can be used for a sufficient schedulability analysis.

In algorithm 4 the new superposition analysis for the event stream model is given. The only difference to the analysis for the periodic model is the inclusion of the event streams.

Algorithm 5 is the corresponding algorithm for calculating the capacity for the event stream model.

If we have a more complex capacity function than $\chi(\Delta t) = \Delta t$ it is necessary to consider additionally those test intervals at which the rate of the additional available execution time changes. A capacity stream can describe such functions. The event stream model is not capable to model complex capacity functions. We will give in chapter 7 a more advanced event model, the hierarchical event streams, that will be able to handle even complex capacity functions.

3.4. Proofs

In the following we will proof the correctness and the approximation characteristic of the superposition approach. We will prove therefore the theorems of the previous sections. It is sufficient to do this proof for the event stream model, as the periodic task model is only a special case of it. A periodic task τ with $\tau = (p, c^+, d)$ can be modeled also by using the event stream $\Theta_\tau = \{(p,0)\}$ so $\tau = (\Theta_\tau, c_\tau^+, d_\tau)$ and a periodic task τ' having a jitter ($\tau' = (p, j, c^+, d)$) can be modeled using $\Theta_{\tau'} = \{(\infty,0),(p,p-j)\}$ so $\tau' = (\Theta_{\tau'}, c_{\tau'}^+, d_{\tau'})$. First it is necessary to prove the condition of lemma 3.1.3. For every possible interval Δt the approximated demand bound function $\delta'(\Delta t,\Gamma,k)$ has to meet or exceed the exact demand bound function $\delta(\Delta t,\Gamma)$.

LEMMA 3.4.1. *If the approximated demand bound function of each event element* θ *of the event stream* Θ *of each tasks* τ *of a task set* Γ $(\delta'(\Delta t,\theta,\tau,k))$ *is larger or equal than the exact demand bound function of this event element* θ $(\delta(\Delta t,\theta,\tau,k))$*, the approximated demand bound function of the task set* Γ $(\delta'(\Delta t,\Gamma))$ *is also larger or equal than the exact demand bound function of the task set* $(\delta(\Delta t,\Gamma))$.

$$\forall\tau\in\Gamma\wedge\forall\theta\in\Theta_\tau \quad : \quad \delta'(\Delta t,\theta,\tau,k) \geq \delta(\Delta t,\theta,\tau) \quad \Rightarrow \quad \delta'(\Delta t,\Gamma,k) \geq \delta(\Delta t,\Gamma)$$

Algorithm 5 Dimensioning a processing element (event stream model)

Algorithm DimProcEvent
Given: task set Γ, k
$testlist := \{\}$
$U_\Gamma = \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau} \frac{c_\tau^+}{p_\theta}$
$C_{result} := U_\Gamma$
$\Delta t_{old} := 0$; $\delta' := 0$
$U_{ready} := 0$
$\forall \tau \in \Gamma \forall \theta \in \Theta_\tau$: ADD $te = (NextInt(0,\theta),\theta)$ TO $testlist$
WHILE ($testlist \neq \{\}$ AND $\Delta t < \Delta t_{max}$)
  Δt :=NEXT INTERVAL OF $testlist$
  θ :=EVENT STREAM ELEMENT BELONGING TO Δt
  $\tau := \tau_\theta$
  REMOVE $(\Delta t, \tau)$ FROM $testlist$
  $\delta' := \delta' + c_\tau^+ + (\Delta t - \Delta t_{old}) U_{ready}$
  $C_{act} := \frac{\delta'}{\Delta t}$
  IF ($C_{act} > C_{result}$)
    $C_{result} := C_{act}$
    $\Delta t_{max} := \frac{\sum_{\forall \tau' \in \Gamma \forall \theta' \in \Theta_{\tau'}} \left(1 - \frac{d_{\tau_{\theta'}}}{p_{\theta'}}\right) c^+_{\tau_{\theta'}}}{C_{act} - U_\Gamma}$
  END IF

  IF ($\Delta t < (d_\tau + p_\theta k)$)
    ADD $\Delta t + p_\theta, \tau$ TO $testlist$
  ELSE
    $U_{ready} := U_{ready} + \frac{c_\tau^+}{p_\theta}$
  END IF
  $\Delta t_{old} := \Delta t$
END WHILE
$\Rightarrow C_{result}$

PROOF. The proof for this lemma follows directly out of the definition 3.3.1 of the approximated demand bound function of a task set:

$$\forall \tau \in \Gamma \wedge \forall \theta \in \Theta_\tau \quad : \quad \delta'(\Delta t, \theta, \tau, k) \geq \delta(\Delta t, \theta, \tau)$$

$$\begin{aligned}\delta'(\Delta t, \theta_{1,\tau_1}, \tau_1, k) + ... + \delta'(\Delta t, \theta_{k,\tau_1}, \tau_1, k) + \\ \delta'(\Delta t, \theta_{1,\tau_2}, \tau_2, k) + ... + \delta'(\Delta t, \theta_{l,\tau_n}, \tau_n, k) \geq & \delta(\Delta t, \theta_{1,\tau_1}, \tau_1, k) + ... + \delta(\Delta t, \theta_{k,\tau_1}, \tau_1, k) + \\ & \delta(\Delta t, \theta_{1,\tau_2}, \tau_2, k) + ... + \delta(\Delta t, \theta_{l,\tau_n}, \tau_n, k)\end{aligned}$$

$$\delta'(\Delta t, \theta, \tau, k) \geq \delta(\Delta t, \theta, \tau)$$

Both complete demand bound functions, the exact and the approximated one, consist of the same set of elements either in its exact or in its approximated form. They are the sum of these elements. Each element of the both sums has one and only one corresponding element for the other sum. Both sums have therefore the same size, e.g the same number of elements. If a sum has the same size then another sum and each of its elements is equal or

larger than its corresponding element of the other sum, than the sum is also equal or larger than the other sum. □

Therefore we can concentrate our considerations in the following on the approximated demand bound function of an event element.

LEMMA 3.4.2. *The exact demand bound function $\delta(\Delta t, \theta, \tau)$ is for any event element θ, any interval Δt and any value of k equal or smaller than the approximated demand bound function $\delta'(\Delta t, \theta, \tau, k)$.*

PROOF. With regard to the definition of the approximated demand bound function given in definition 3.3.1 we have to distinguish two cases. The proof for the first case, covering all intervals being evaluated exactly ($\Delta t < a_\theta + d_\tau + kp = \Delta t_\theta$), follows directly out of definition 3.1.1. The second case covers all intervals that are equal or larger than Δt_θ and that are therefore approximated ($\Delta t \geq \Delta t_\theta$). The proof for these cases is given by:

$$\begin{aligned}
\delta'(\Delta t, \theta, \tau, k) &= \delta(\Delta t_{\theta,k}, \tau) + \frac{c_\tau^+}{p_\theta}(\Delta t - \Delta t_{\theta,k}) \\
&= \left(\frac{\Delta t_{\theta,k} - a_\theta - d_\tau}{p_\theta} + 1\right) c_\tau^+ + \frac{c_\tau^+}{p_\theta}\left(\Delta t - \Delta t_{\theta,k}\right) \\
&= \left(\frac{\Delta t_{\theta,k} - a_\theta - d_\tau + \Delta t - \Delta t_{\theta,k}}{p_\theta} + 1\right) c_\tau^+ \\
&= \left(\frac{\Delta t - a_\theta - d_\tau}{p_\theta} + 1\right) c_\tau^+ \\
&\geq \left\lfloor \frac{\Delta t - a_\theta - d_\tau}{p_\theta} + 1 \right\rfloor c_\tau^+ = \delta(\Delta t, \theta, \tau)
\end{aligned}$$

As consequence the approximated demand bound function is always greater or equal than the exact demand bound function. □

The next step is to prove that it is sufficient to check the approximated demand bound function for the remaining test intervals to proof schedulability. We have to distinguish between two cases. The first case is that the capacity function is equal to the intersection ($\chi(\Delta t) = \Delta t$).

LEMMA 3.4.3. *Let $\Delta t_i, \Delta t_j$ be two consecutive test intervals for the approximated demand bound function δ' of the task set Γ. If there exists an interval $\Delta t'$ with $\Delta t_i < \Delta t' < \Delta t_j$ and $\delta(\Delta t', \Gamma) > \Delta t'$ and $U_\Gamma \leq 100\%$ then the value of the approximated demand bound function exceeds the available execution time also at interval Δt_i, therefore $\delta'(\Delta t_i, \Gamma, k) > \Delta t_i$.*

In the case that the exact demand exceeds the available execution time at some interval $\Delta t'$ and this interval $\Delta t'$ is not a test interval for an approximated demand bound function, this approximated demand bound function will also exceed the available execution time for the largest interval being smaller than $\Delta t'$ and being a test interval of the approximated demand bound function. The situation is visualized in figure 3.4.1.

PROOF. The proof for lemma 3.4.3 is given in the following.

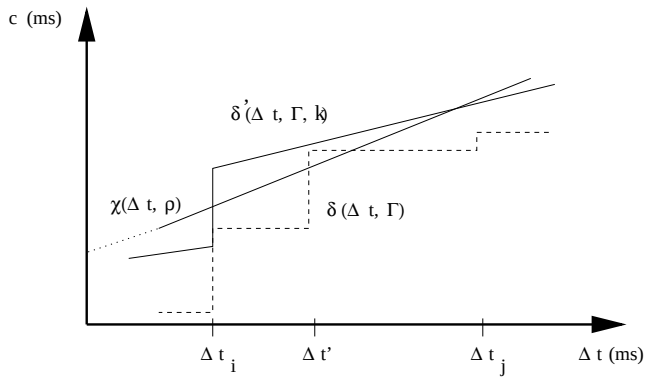

FIGURE 3.4.1. Visualization of lemma 3.4.3

Let us assume that $\Delta t'$ is a test interval of the non-approximated demand bound function with $\delta(\Delta t', \Gamma) > \Delta t'$ and that Δt_i is the largest test interval being smaller than $\Delta t'$ for a certain approximative demand bound function δ'.

For the approximated demand bound function δ' the event stream Θ_τ of each task is split up into those event elements being considered exactly Θ_τ^e and those event elements being approximated Θ_τ^a with regard to interval Δt_i. The event elements of a task can be distributed in both groups. We know that $\Theta_\tau = \Theta_\tau^e \cup \Theta_\tau^a$ and $\Theta_\tau^e \cap \Theta_\tau^a = \emptyset$, therefore

$$\delta(\Delta t, \tau) = \sum_{\forall \theta \in \Theta_\tau^e} \delta(\Delta t, \theta, \tau) + \sum_{\forall \theta \in \Theta_\tau^a} \delta(\Delta t, \theta, \tau)$$

Let us rewrite the demand bound function for $\Delta t'$ using the split task set:

$$\sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^e} \delta(\Delta t', \theta, \tau, k) + \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^a} \delta(\Delta t', \theta, \tau) > \Delta t'$$

Because due to the condition in lemma 3.4.3 there is no test interval between Δt_i and $\Delta t'$ with regard to the tasks considered exactly, so we know that:

$$\forall \theta \in \Theta_\tau^e \quad : \quad \delta(\Delta t', \theta, \tau) = \delta(\Delta t, \theta, \tau)$$

We also know from lemma 3.4.2 that the approximated demand bound functions have always an equal or larger value than the exact demand bound functions ($\delta'(\Delta t, \theta, \tau, k) \geq \delta(\Delta t, \theta, \tau)$). With these conditions the inequality can be rewriten:

$$\sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^e} \delta'(\Delta t_i, \theta, \tau, k) + \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^a} \delta'(\Delta t', \theta, \tau, k) > \Delta t'$$

By using definition 3.1.2 of the approximated demand bound function the demand of the approximated event stream elements can be split. As the capacity function is equal to the intersection we only allow utilizations lower than the available capacity of 100% here. Therefore

$$\sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^a} \frac{c_\tau^+}{p_\theta} \leq \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau} \frac{c_\tau^+}{p_\theta} \leq 1$$

With this condition we achieve:

$$\delta'(\Delta t_i, \Gamma, k) + \Delta t' - \Delta t_i > \Delta t'$$

$$\delta'(\Delta t_i, \Gamma, k) > \Delta t_i$$

In case that the demand bound function exceeds the intersection, which models the available capacity, for any interval Δt the approximated demand bound function also exceeds the intersection at the test interval before Δt. □

The proof of the algorithms 2, 3 and 5 which calculate the minimum possible capacity of a resource requires a different lemma than lemma 3.4.3. We have to proof for these algorithms that the skipped test intervals cannot lead to a higher required capacity than the previous considered test-interval. Also is $U \leq 1$ no longer a valid assumption.

LEMMA 3.4.4. *Let $\Delta t_i, \Delta t_j$ be two consecutive test intervals for the approximated demand bound function δ' of the task set Γ. Let C_r be a required capacity with $C_r \geq U_\Gamma$. If there exists an interval $\Delta t'$ with $\Delta t_i < \Delta t' < \Delta t_j$ and $\delta(\Delta t', \Gamma) = C_r \Delta t'$ then the value of the approximated demand bound function reaches at least the same required capacity also at interval Δt_i, therefore $\delta'(\Delta t_i, \Gamma, k) \geq C_r \Delta t_i$.*

PROOF. The proof for this lemma is quite similar to the proof for lemma 3.4.3. We have

$$\sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^e} \delta(\Delta t', \theta, \tau, k) + \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^a} \delta(\Delta t', \theta, \tau) = C_r \Delta t'$$

Following the proof of above we achieve

$$\sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^e} \delta'(\Delta t_i, \theta, \tau, k) + \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^a} \delta'(\Delta t', \theta, \tau, k) \geq C_r \Delta t'$$

As the utilization does not exceed C_r we can assume

$$\sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau^a} \frac{c_\tau^+}{p_\theta} \leq \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau} \frac{c_\tau^+}{p_\theta} \leq C_r$$

and achieve with this condition

$$\delta'(\Delta t_i, \Gamma, k) + \Delta t' - \Delta t_i \geq C_r \Delta t'$$
$$\delta'(\Delta t_i, \Gamma, k) \geq C_r \Delta t_i$$

The capacity required for a skipped test interval is also required for the previous considered test interval □

General capacity functions are considered in chapter 7 where we present models for these functions.

Let us now consider the proof of the approximation condition:

LEMMA 3.4.5. *The relative error between the approximated demand bound function and the exact demand bound function is bounded by $\frac{1}{k}$.*

$$\frac{\delta'(\Delta t, \Gamma, k) - \delta(\Delta t, \Gamma)}{\delta(\Delta t, \Gamma)} \leq \frac{1}{k}$$

First we will determine the maximum absolute approximation error.

LEMMA 3.4.6. *For an event stream element θ the maximum difference between any approximated demand bound function $\delta'(\Delta t, \theta, \tau, k)$ and the corresponding exact demand bound function $\delta(\Delta t, \theta, \tau)$ is bounded by one time the execution time of the task $c^+_{\tau_\theta}$.*

PROOF. Let, without loss of generality, Δt be any interval and k be any number of exactly analyzed test intervals:

$$\begin{aligned}
\delta'(\Delta t, \theta, \tau, k) - \delta(\Delta t, \theta, \tau) &= \delta(\Delta t_{\theta,k}, \theta) + \frac{c^+_\tau}{p_\theta}(\Delta t - \Delta t_{\theta,k}) - \delta(\Delta t, \theta) \\
&= \frac{\Delta t_{\theta,k} - a_\theta - d_\tau}{p_\theta} c^+_\tau + \frac{c^+_\tau}{p_\theta}(\Delta t - \Delta t_{\theta,k}) - \left\lfloor \frac{\Delta t - a_\theta - d_\tau}{p_\theta} \right\rfloor c^+_\tau \\
&= \frac{c^+_\tau}{p_\theta}(\Delta t - a_\theta - d_\tau) - \left\lfloor \frac{\Delta t - a_\theta - d_\tau}{p_\theta} \right\rfloor c^+_\tau \\
&\leq \frac{c^+_\tau}{p_\theta}(\Delta t - a_\theta - d_\tau) - \left\lceil \frac{\Delta t - a_\theta - d_\tau}{p_\theta} - 1 \right\rceil c^+_\tau \\
&\leq \frac{c^+_\tau}{p_\theta}(\Delta t - a_\theta - d_\tau) - \left(\frac{\Delta t - a_\theta - d_\tau}{p_\theta} - 1 \right) c^+_\tau \\
&\leq c^+_\tau
\end{aligned}$$

□

Therefore the maximum error of an approximative demand-bound-function $\delta'(\Delta t, \theta, k)$ is limited to one time the execution time of the task c_{τ_θ}. It only applies if the task is approximated, therefore if $\Delta t \geq k p_\theta + d_{\tau_\theta} + a_\theta$. Using this result the error of the approximated demand bound function $\delta'(\Delta t, \Gamma, k)$ and therefore $\delta'(\Delta t, \Gamma, k)$ can be bounded too.

LEMMA 3.4.7. *The error of the approximated demand bound function is bounded by:*

$$\delta'(\Delta t, \Gamma, k) - \delta(\Delta t, \Gamma) \leq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t > \Delta t_{\theta,k}}} c^+_\tau$$

PROOF. The proof for the lemma follows out of the previous statements. □

LEMMA 3.4.8. *For an interval Δt the demand bound function has at least the value:*

$$\delta(\Delta t, \Gamma) \geq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t > \Delta t_{\theta,k}}} k c^+_\tau$$

PROOF.

$$\begin{aligned}
\delta(\Delta t, \Gamma) &\geq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t > \Delta t_{\theta,k}}} \delta(\Delta t_{\theta,k}, \theta, \tau) \\
&\geq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t > \Delta t_{\theta,k}}} \left\lfloor \frac{k p_\theta + d_\tau - d_\tau}{p_\theta} + 1 \right\rfloor c^+_\tau \\
&\geq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t > \Delta t_{\theta,k}}} (k+1) c^+_\tau \\
&\geq \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t > \Delta t_{\theta,k}}} k c^+_\tau
\end{aligned}$$

With these two bounds it is possible to prove theorem 3.1.4, in which it is assumed that if a task set Γ is schedulable on a resource ρ_1, it is guaranteed for the approximative analysis to return "schedulable" for a resource ρ_2 with a slightly higher capacity than ρ_1:

$$\delta(\Delta t, \Gamma) \leq \chi(\Delta t, \rho_1)$$
$$\Rightarrow \delta'(\Delta t, \Gamma, k) \leq \chi(\Delta t, \rho_2)$$

The relationship between the capacities is given by

$$\chi(\Delta t, \rho_2) = \chi(\Delta t, \rho_1) + \frac{1}{k}\chi(\Delta t, \rho_1)$$

We have $\delta(\Delta t, \Gamma) \leq \chi(\Delta t, \rho_1)$. It follows:

$$\delta(\Delta t, \mathscr{S}) \leq \chi(\Delta t, \rho_1)$$
$$\delta(\Delta t, \mathscr{S}) + \frac{1}{k}\delta(\Delta t, \mathscr{S}) \leq \chi(\Delta t, \rho_1) + \frac{1}{k}\delta(\Delta t, \mathscr{S})$$

Using definition 3.3.1 we get:

$$\delta'(\Delta t, \mathscr{S}, k) \leq \chi(\Delta t, \rho_1) + \frac{1}{k}\delta(\Delta t, \mathscr{S})$$
$$\delta'(\Delta t, \mathscr{S}, k) \leq \chi(\Delta t, \rho_1) + \frac{1}{k}\chi(\Delta t, \rho_1)$$
$$\delta'(\Delta t, \mathscr{S}, k) \leq \chi(\Delta t, \rho_2)$$

The relative difference between the capacities of ρ_1 and ρ_2 is simply:

$$\frac{\chi(\Delta t, \rho_2) - \chi(\Delta t, \rho_1)}{\chi(\Delta t, \rho_1)} = \frac{\chi(\Delta t, \rho_1) + \frac{1}{k}\chi(\Delta t, \rho_1) - \chi(\Delta t, \rho_1)}{\chi(\Delta t, \rho_1)} \leq \frac{1}{k}$$

which is equal to the maximum difference between the demand bound function $\delta(\Delta t, \Gamma)$ and the approximated demand bound function $\delta'(\Delta t, \Gamma, k)$ and therefore to the error:

$$\frac{\delta'(\Delta t, \Gamma, k) - \delta(\Delta t, \Gamma)}{\delta(\Delta t, \Gamma)} \leq \frac{\sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t > \Delta t_{\theta,k}}} c_\tau^+}{\sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \theta \in \Theta_\tau \\ \Delta t > \Delta t_{\theta,k}}} k c_\tau^+} = \frac{1}{k}$$

The error of the approximation for the complete task set is always bounded by $\frac{1}{k}$. □

3.5. Approximation error

It is necessary to investigate the term of error used in the approximation schemes more closely. This term is not as clear as it seems on the first view.

3.5.1. General remarks. The error somehow depends on the question to solve. For example the backpacking-problem has the question whether it is possible to pack items with a certain value in a backpack with a given size. For the schedulability analysis the question is whether it can be guaranteed for a given task set to meet all its deadlines while executed on a processing element with a certain capacity. These questions are true/false decisions.

Each of these questions can be transferred into a corresponding optimization problem. For the backpacking problem, the goal of the optimization algorithms would be to find

either the maximum value of those goods which could be packed into a given backpack or the minimum backpack which can accept a set of goods with a given value.

Each of these optimization problems leads to a possible approximation scheme with a different kind of error. For the first problem the error would be measured in percentage of the value of the goods, for the second problem the error would be measured in the size of the backpack.

For example, if the optimal solution allows goods for 1000 $ an approximation scheme with an error of 1% has to find a solution with a value of at least 990 $. In the second case the approximation scheme would allow a slightly larger backpack than the optimal. The difference between these backpacks is bounded by the error.

Those are two different questions leading to different algorithms for the approximation (of cause only if such approximations exists).

For our schedulability problem we cannot vary the length of the tasks. Perhaps it would be possible to vary the length of the deadlines, but we do not have a measurement to compare the deadlines of different task sets.

In soft real-time systems an optimization criterion could be to minimize the maximum latency or the average latency of tasks. But for hard real-time systems we do not allow any latency at all.

So we have as only adjustable parameter the capacity of the chosen processor. The optimization question is to find the processor with the minimum capacity for which it is possible to guarantee the deadlines of all tasks in the task set. This leads to an approximation scheme in which the error is measured in the size of the capacity of the processor.

An algorithm fulfilling this approximation scheme will guarantee that a task set wrongly classified to be “not schedulable” on some processors is guaranteed to be “not schedulable” on another processor with a slightly lower capacity. This capacity of the other processor is given by the previous capacity and the approximation error.

This definition of the error is similar to the definition given by Axelsson [10] p. 82 ff. He uses the term “minimum required speed-up”. This minimum required speed-up is defined as how much additional capacity for the processor is needed to reach schedulability.

3.5.2. Approximation error for schedulability analysis. For the schedulability analysis two different concepts for the approximation errors exists.

The first concept is related to the time in the system and was proposed by Chakraborty et al. [38]. The maximum test interval, a value of time, is divided into equally distributed test intervals. Every demand occurring between two of these test intervals is considered as having occurred either in the smaller one or the larger one of these test intervals.

Considering the demand as having occurred in the smaller test interval leads to a pessimistic analysis, which means an overestimation of the required resources or an underestimation of the response times. Considering the demand as having occurred in the larger test interval leads to an optimistic analysis, which means an underestimation of the response times or the necessary resources. Note, that this overestimation and underestimations are bounded by the approximation error. An analysis having a difference of 10 ms between two consecutive test intervals can lead to deadline misses of respectively 10 ms at most.

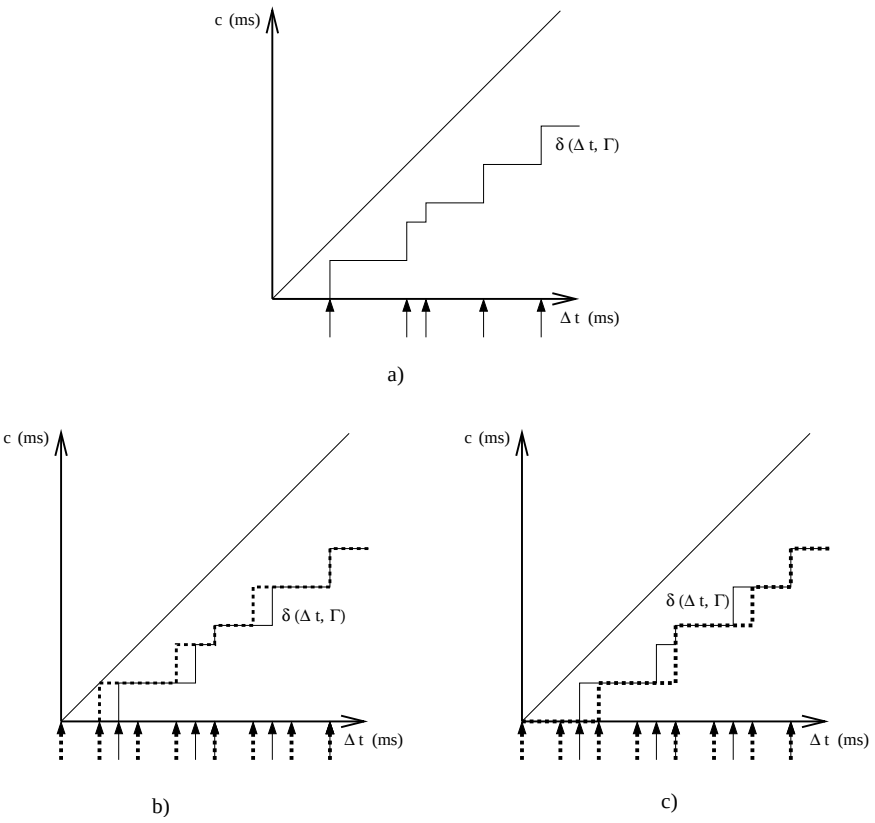

FIGURE 3.5.1. Approximation related to the time
a) Exact case b) pessimistic approximation c) optimistic approximation

Figure 3.5.1 shows an example for this approximation. In a) the exact demand bound function and its test intervals are given. In the Figure b) the dotted line shows the same demand bound function using a pessimistic approximation with equally distributed test intervals. Any value of the approximated demand bound function occurs at an equal or earlier time than the same value of the exact demand bound function.

The approximated demand bound function is still feasible in the example but meets the capacity for the second test interval. In c) an optimistic approximation of the same demand bound function is shown. The values for the approximated demand bound function occur later or at the same time than the same values of the exact demand bound function. This can lead to deadline misses as the response of a task can end a bit later than predicted by the analysis.

The distance between two consecutive test intervals determines the error of the time related approximation. This distance depends on two conditions, the chosen maximum number of allowed test intervals and therefore the chosen effort for the analysis on the one side, and the length of the maximum test interval on the other side.

The maximum test interval depends not on the number of tasks in a task set but on the parameter of these tasks and on the utilization of the complete task set. A high utilization or large periods leads to a long maximum test interval and therefore to a long distance

between consecutive test intervals. This distance is equal to the error so a long maximum test interval leads to a large error.

The size of the maximum test interval depends of the parameters of the task in the task set. For this reason, the error of the time related approximation also depends on these parameters. The problem is that this error is the same absolute value for all tasks in the task set independently whether they are small or large. Tasks of very different sizes in one task set can lead to a large maximum test interval. For the smaller tasks in the task set the error can be quite large compared to their periods and deadlines.

For example having a task set with a task τ_1 with $d_{\tau_1} = 10$ ms and tasks with periods of 1,000 ms leading to a maximum test interval of 2,500 ms. Assuming an allowed effort of 1,000 test intervals leads to an error of 0.1% and an absolute difference between the test intervals of 2.5 ms. Despite that the error seems to be very small for the complete task set, 2.5 ms is 25% of the deadline of τ_1. The task is either allowed to miss its deadline by 2.5 ms so to end at 12.5 ms or the task has to respond within 7.5 ms to guarantee its deadline using this approximation. The error of this approach is the maximum additional demand allowed to occur between any two test intervals.

The second kind of approximation error is the one used in our approach. The approximation is bounded by a fraction of the capacity, not a value of time. The value of the approximated demand bound function is always equal or larger than the value of the exact demand bound function. We only consider the pessimistic case for this approximation. The resulting performance, especially for the adaptive approximation, which we will introduce in chapter 4, is so satisfactory that an optimistic approach with its potential deadline misses is not necessary. The error of our approximation does not, other than the error of the time related approximation, depend on the parameters of the task set.

By definition of approximations the type of error of the approximation need to be measured with the same type of value as for the optimization. In our approach the error is of type “capacity of the resource” and the optimization wants to minimize this capacity. For the goal to minimize the latency of the task an error of time would be adequate. For achieving a small error a price has to be paid in increasing the necessary effort. The advantage is that the approximation allows a trade-off between the degree of exactness and the required evaluation effort.

We have an error that can be chosen and reduced as near as desired to the exact solution. To prove that the superposition algorithm is a *fully polynomial time approximation scheme* we only have to show that the superposition algorithm has a polynomial complexity with regard to the number of tasks and the chosen approximation error.

3.6. Complexity

The complexity of the schedulability problem for the synchronous uni-processor case is unknown. The best available exact analysis has a pseudo-polynomial complexity if the utilization is bound by a value smaller than 100% [17, 84]. One example for such an algorithm is the processor demand criterion given in [19].

The complexity of our algorithm 4 depends, same as for each other approximation, on two values: The number of tasks of the considered task set and the chosen approximation error. Let n be the number of tasks in the task set and $\varepsilon = \frac{1}{k}$ be the chosen approximation error. The consideration of a single test interval needs several operations with linear complexity (addition, multiplication, comparison) and one operation for inserting a new element into a sorted list. Due to the required priority queue the overall complexity for considering a single test interval is $O(n \log n)$. The maximum number of test intervals that have to be considered during one run of the algorithm can be bound by the number of tasks multiplied with the maximum number of exactly considered test intervals for each task. This number is $k = \frac{1}{\varepsilon}$ and therefore depends directly on the approximation error. Calculating the maximum test interval, with exception of the busy period which we therefore have not used, and the initialization requires also not more than $O(n \log n)$. Therefore the overall complexity of the superposition algorithm can be bounded by $O(n \log n \frac{1}{\varepsilon})$.

The complexity of our algorithm depends polynomial on the number of tasks and on the chosen error. It is a fully polynomial time approximation scheme.

3.7. Comparison to related work

Comparing the superposition algorithm to previous related work leads to interesting results. In the following chapter we will discuss the relation of the superposition algorithms to the best sufficient analysis and the best exact analysis and prove that both are only special cases of the superposition analysis.

3.7.1. Best sufficient analyses. In this chapter we will show and prove the relationship of the superposition approach to the best available sufficient test for EDF scheduling, that was proposed by Devi in [46].

We will show that the superposition approach is a generalization of the previously existing sufficient schedulability analysis for EDF. Liu and Layland [88] proved that for a simple periodic task model with all tasks having a deadline equal to their period all task sets with a utilization equal or lower 100% are feasible ($\sum_{\tau \in \Gamma} \frac{c_\tau^+}{p_\tau} \leq 1$). As the response times for a task set with a utilization larger than 100% becomes infinitely large, this test is also necessary. The task model only allows tasks activated by periodic events and having a relative deadline equal to the period of the tasks. The advantage of this analysis is the linear complexity $O(n)$.

For more generalized periodic models, having deadlines smaller than the periods of the tasks or a more powerful event model, this test is no longer sufficient. A good sufficient analysis with also linear complexity was developed [90].

THEOREM 3.7.1. *[90] For a task set with tasks activated by periodic events with a period p and having a relative deadline d, $\sum_{\tau \in \Gamma} \frac{c_\tau^+}{min(p_\tau, d_\tau)} \leq 1$ is a sufficient schedulability analysis.*

PROOF. Given is a task set $\Gamma = \{\tau_1, ..., \tau_n\}$. Let us assume a task set Γ' with $\Gamma' = \{\tau'_1, ..., \tau'_n\}$, $p_{\tau'_i} = \min(p_{\tau_i}, d_{\tau_i})$, $d_{\tau'_i} = \min(p_{\tau_i}, d_{\tau_i})$, $c^+_{\tau'_i} = c^+_{\tau_i}$. From the theorem we know

that

$$\sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{\min(p_\tau, d_\tau)} \leq 1$$

$$\Rightarrow \sum_{\forall \tau' \in \Gamma'} \frac{c_{\tau'}^+}{p_{\tau'}} \leq 1$$

So Γ' is schedulable if the condition of the theorem holds for Γ. Without loosing the schedulability we can enlarge $p_{\tau_i'}$ to p_{τ_i} and $d_{\tau_i'}$ to d_{τ_i}. So if Γ' is schedulable Γ is also schedulable. □

Based on this approach Devi [46] has developed a new sufficient analysis. It is a variation of the test above, but has to check one in-equation for each task.

THEOREM 3.7.2. *([46]) A task set Γ, arranged in order of non-decreasing relative deadlines is feasible using EDF scheduling if $n = |\Gamma|$ and*

$$\forall k \leq n \mid \left(\sum_{i=1}^{k} \frac{c_{\tau_i}^+}{p_{\tau_i}} + \frac{1}{d_{\tau_k}} \sum_{j=1}^{k} \frac{p_{\tau_j} - \min(p_{\tau_{ij}}, d_{\tau_j})}{p_{\tau_j}} c_{\tau_j}^+ \leq 1 \right)$$

PROOF. (See [46]) □

Despite that it is necessary for the analysis to test one in-equation for each task, it still has a linear complexity. For all task sets this analysis leads to equal or higher bounds than the previous sufficient analysis.

LEMMA 3.7.3. *A task set recognized as schedulable by the analysis proposed in lemma 3.7.1 is also recognized as schedulable by the analysis of Devi [46] (theorem 3.7.2).*

PROOF. (See [46]) □

With the superposition approximation we can generalize this result even further and prove that the superposition is a generalization of the test of Devi.

THEOREM 3.7.4. *The task sets that are recognized as schedulable by the analysis proposed in lemma 3.7.2 are also recognized as schedulable by using the superposition analysis with the approximative demand bound function $\delta'(\Delta t, \Gamma, k)$ with $k = 1$.*

PROOF. For the analysis with $\delta'(\Delta t, \Gamma, 1)$ only one test interval needs to be considered for each task $\tau \in \Gamma$. In general we have

$$\delta'(\Delta t, \Gamma, 1) = \sum_{\forall \tau \in \Gamma} \begin{cases} \left(\frac{\Delta t - d_\tau}{p_\tau} \right) c_\tau^+ + \delta(d_\tau, \tau) & \Delta t \geq d_\tau \\ 0 & \Delta t < d_\tau \end{cases}$$

$$\sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq \Delta t}} \left(\frac{c_\tau^+}{p_\tau} (\Delta t - d_\tau) + c_\tau^+ \right)$$

$$= \sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq \Delta t}} c_\tau^+ + \Delta t \sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq \Delta t}} \frac{c_\tau^+}{p_\tau} - \sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq \Delta t}} \frac{d_\tau}{p_\tau} c_\tau^+$$

The test of theorem 3.7.2 can be transformed:

$$\begin{aligned} d_{\tau_k} &\geq d_{\tau_k} \sum_{i=1}^{k} \frac{c_{\tau_i}^+}{p_{\tau_i}} + \sum_{i=1}^{k} \left(\frac{p_{\tau_i} - \min(p_{\tau_i}, d_{\tau_i})}{p_{\tau_i}} \right) c_{\tau_i}^+ \\ &\geq d_{\tau_k} \sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq d_{\tau_k}}} \frac{c_\tau^+}{p_\tau} + \sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq d_{\tau_k}}} \left(\frac{p_\tau - d_\tau}{p_\tau} \right) c_\tau^+ \\ &\geq \sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq d_{\tau_k}}} c_\tau^+ + d_{\tau_k} \sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq d_{\tau_k}}} \frac{c_\tau^+}{p_\tau} - \sum_{\substack{\forall \tau \in \Gamma \\ d_\tau \leq d_{\tau_k}}} \frac{d_\tau}{p_\tau} c_\tau^+ \end{aligned}$$

The test of Devi has to be performed one time for each task $\tau_k \in \Gamma$. The calculation for $\delta'(\Delta t, \Gamma, 1)$ has to be done also one time for each task at the first deadline d_{τ_k} of each task. Using $\delta'(d_{\tau_k}, \Gamma, 1) \leq d_{\tau_k}$ as test leads to an equal calculation for both tests. □

In other words, the test by Devi is only a special case of the superposition approach. As we have already seen the processor demand criterion is also only a special case of the superposition approximation, the case with $k = \infty$. Therefore the superposition approximation bridges the gap between the fast but only sufficient test of Devi and the accurate but slow processor demand criterion and therefore between the best sufficient and the best exact analysis and combines them in one consistent approach.

3.7.2. Asynchronous task sets. The problem of schedulability analysis for asynchronous task sets, e.g. task sets in which the tasks can have different offsets, for EDF scheduling is CO-NP hard [18]. In [106, 104, 105] a sufficient schedulability analysis was proposed for these asynchronous task sets scheduled with EDF. This approaches requires a repeated use of the processor demand criterion. In [68] it was shown that by a combination of the approach given by Pellizzoni and Lipari in [104] with the introduced approximation and the dynamic approximation proposed in the next chapter, a significant reduction of the run-time of the analysis could be achieved.

CHAPTER 4

Adaptive schedulability tests

Approximative schedulability tests are only sufficient. Not all schedulable systems are recognized as such by these analyses. Even that the degree of exactness can be chosen does not help. The selection of the exactness has to be done in advance. Choosing a high degree of exactness can lead to a long execution time of the analysis whereas choosing a low degree might result in a failure to classify correctly the systems in question.

But the proposed approximative superposition test can be used to achieve fast exact schedulability test algorithms. The idea is to adapt the error dynamically on the hardness of the problem. Systems being classified correctly by an inexact approximation should be evaluated fast by the new analysis and only systems needing a high degree of exactness for their correct classification will need a long evaluation time. Therefore the proposed algorithm uses different levels of approximation, starting with an inexact but fast approximation. In this chapter we will, based on the proposed approximation algorithms, introduce exact feasibility analysis for dynamic priority systems.

We will propose two exact feasibility analyses, the dynamic error analysis and the all approximation analysis. We will consider the period task model with jitter and the event stream model.

4.1. Dynamic error analysis

The idea behind the dynamic error analysis is to use different levels of approximation having different degrees of exactness. The test starts at an approximation level with a low degree of exactness resulting in a fast schedulability test. The test switches to a level with a higher degree of exactness if it fails at the lower level.

The example in figure 4.1.1 shows an exact demand bound function. In figure 4.1.2 the dynamic test for the same function is shown. The test starts with an approximation level of one test interval for each event stream element or task. The first test interval is

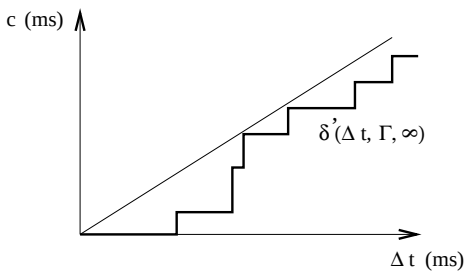

FIGURE 4.1.1. Exact demand bound function

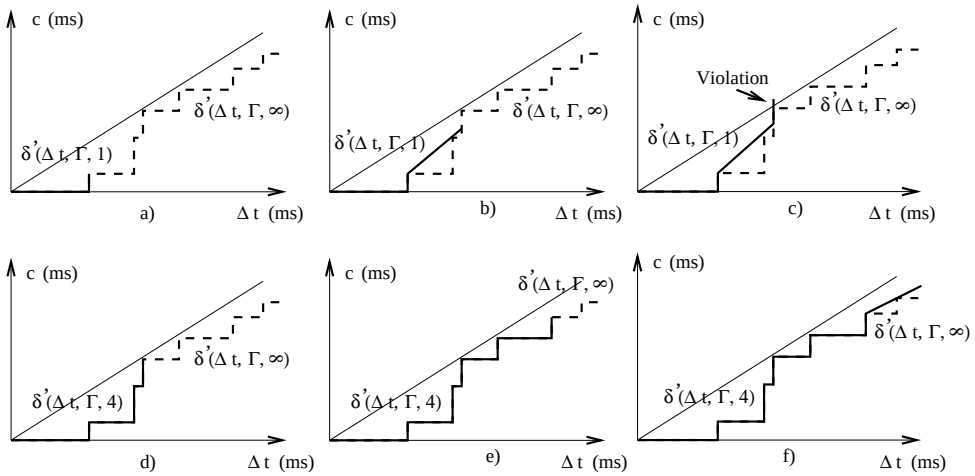

FIGURE 4.1.2. Graphical visualization for an example of the dynamic error test

considered exactly (figure 4.1.2 a) and then the approximation starts for this task (figure 4.1.2 b). Considering the first test interval for the second task results in a violation due to the previous approximation of the first task (figure 4.1.2 c). Therefore the approximation level is raised (to two test intervals for each task) and the value for the first test interval of the second task is recalculated (figure 4.1.2 d) . The test continues with the new approximation level (figure 4.1.2 e) and now does not lead to any more violation (figure 4.1.2 f), so the test returns “schedulable”.

It is not necessary to recalculate the value of the demand bound function $\delta(\Delta t, \Gamma, k')$ for the new approximation level. It is only necessary to calculate the difference to the previous value of $\delta(\Delta t, \Gamma, k)$. This difference depends on those tasks only for which an approximated value is included in the total value on the previous level and which are no longer approximated on the new level.

LEMMA 4.1.1. *Calculating the demand bound function for a new approximation level*

Let $k_o = \frac{1}{\varepsilon_{old}}, k_n = \frac{1}{\varepsilon_{new}}$ be two approximation levels and Γ_{diff} be the set of tasks that are not approximated on level ε_{new} and have previously be approximated on level ε_{old} with regard to an interval Δt. The demand bound function of the new level can be calculated:

$$\delta'(\Delta t, \Gamma, k_n) = \delta'(\Delta t, \Gamma, k_o) - \left(\sum_{\forall \tau \in \Gamma_{diff}} \left(\frac{\Delta t + j_\tau - d_\tau}{p_\tau} - \left\lfloor \frac{\Delta t + j_\tau - d_\tau}{p_\tau} \right\rfloor \right) \right)$$

or for the event stream model:

$$\delta'(\Delta t, \Gamma, k_n) = \delta'(\Delta t, \Gamma, k_o) - \left(\sum_{\forall \tau \in \Gamma_{diff}} \sum_{\forall \theta \in \Theta_\tau} \left(\frac{\Delta t - a_\theta - d_\tau}{p_\theta} - \left\lfloor \frac{\Delta t - a_\theta - d_\tau}{p_\theta} \right\rfloor \right) \right)$$

PROOF. We need to consider the definition of the approximated demand bound function. It is sufficient to do the proof for the event stream model, as the periodic model is

included in it as special case.

$$\delta'(\Delta t,\Gamma,k_o)-\delta'(\Delta t,\Gamma,k_n) \quad = \quad \sum_{\forall\tau\in\Gamma}\delta'(\Delta t,\tau,k_o)-\sum_{\forall\tau\in\Gamma}\delta'(\Delta t,\tau,k_n)$$

Due to the definition of Γ_{diff} we have the condition:

$$\forall\tau\notin\Gamma_{diff} \quad | \quad \delta'(\Delta t,\tau,k_o)=\delta'(\Delta t,\tau,k_n)$$

Therefore we achieve:

$$\begin{aligned}\delta'(\Delta t,\Gamma,k_o)-\delta'(\Delta t,\Gamma,k_n) &= \sum_{\forall\tau\in\Gamma_{diff}}\delta'(\Delta t,\tau,k_o)-\sum_{\forall\tau\in\Gamma_{diff}}\delta'(\Delta t,\tau,k_n)\\ &= \sum_{\forall\tau\in\Gamma_{diff}}\sum_{\forall\theta\in\Theta_\tau}\frac{\Delta t-a_\theta-d_\tau}{p_\theta}-\sum_{\forall\tau\in\Gamma_{diff}}\sum_{\forall\theta\in\Theta_\tau}\left\lfloor\frac{\Delta t-a_\theta-d_\tau}{p_\theta}\right\rfloor\\ &= \sum_{\forall\tau\in\Gamma_{diff}}\sum_{\forall\theta\in\Theta_\tau}\left(\frac{\Delta t-a_\theta-d_\tau}{p_\theta}-\left\lfloor\frac{\Delta t-a_\theta-d_\tau}{p_\tau}\right\rfloor\right)\end{aligned}$$

□

The problem is whether the whole analysis has to be repeated when switching to a higher approximation level and therefore the previously considered (and skipped) test intervals have to be reconsidered or not. Fortunately this is unnecessary.

THEOREM 4.1.2. *Let interval Δt be a test interval on the approximation level k. Let $\Delta t'$ be the largest test interval on the approximation level $k'<k$ with a size smaller than Δt. If $\delta(\Delta t,\Gamma,k)>\Delta t$ than $\delta(\Delta t',\Gamma,k)>\Delta t'$.*

In other words, if the analysis on the new approximation level would have failed for any test intervals it would have also failed for the previous test interval on a lower approximation level and therefore for one of the considered test intervals. Therefore it is not necessary to reconsider the skipped test intervals.

PROOF. If the analysis fails for an interval Δt using an approximation level k it fails for Δt also with each approximation level $k'<k$. The remaining part of the lemma is equivalent to lemma 3.4.3 therefore the proof is also the same. □

In case of a level switch it is possible to continue the analysis at the considered test interval. It is necessary to complete the *testlist* by the test intervals of those tasks that are no longer approximated due to the change of the approximation level. Let Γ_{next} be again the set of tasks that are no longer approximated on level k_{next} at interval Δt.

LEMMA 4.1.3. *Let $\tau\in\Gamma_{next}$ be a task being approximated at approximation level k_{prev} and being no longer approximated at level k_{next}. $NextInt(\Delta t,\tau)$ gives the smallest test interval for task τ with a size larger than Δt.*

$$NextInt(\Delta t,\tau)=\max\left(d_\tau,\left(\left\lfloor\frac{\Delta t+j_\tau-d_\tau}{p_\tau}\right\rfloor+1\right)p_\tau+d_\tau-j_\tau\right)$$

or for the event stream model with the event element θ:

$$NextInt(\Delta t,\theta)=(\left\lfloor\frac{\Delta t-a_\theta-d_{\tau_\theta}}{p_\theta}\right\rfloor+1)p_\theta+a_\theta+d_{\tau_\theta}$$

PROOF. It is again sufficient to do the proof only for the event stream model. The intervals generated by an event stream element θ are given by

$$\Delta t_{test} = kp_\theta + a_\theta + d_{\tau_\theta} \quad k \in \mathbb{N}$$

For a specific value of k we get:

$$k = \frac{\Delta t_{test} - a_\theta - d_{\tau_\theta}}{p_\theta}$$

Only natural numbers are allowed for k. As it is required for the next larger interval Δt_{test} to the interval Δt being not equal to Δt we get for k:

$$k = \left\lfloor \frac{\Delta t_{test} - a_\theta - d_{\tau_\theta}}{p_\theta} \right\rfloor + 1$$

This leads to the above calculation of $NextInt(\Delta t, \theta)$. □

The dynamic error schedulability test is formulated in algorithm 6. The test starts at test level k_1 in which one test interval is considered for each task. Same as in the original superposition algorithm, the smallest test interval for each task has to be collected in the *testlist*. The algorithm considers the intervals in *testlist* in increasing order. For each of these intervals the approximated demand bound function δ' is calculated and compared with the available computation time for this interval. If the available computation time is equal or larger than the required computation time given by the demand bound function, schedulability is proven with regard to this test interval.

In general the next larger test interval for the task responsible for the checked test interval is added. This last step is skipped in cases where the maximum number of test intervals for this task on the actual approximation level is reached. In this case no more test intervals are considered and therefore added to *testlist* for this task. For compensation it is necessary to add an approximated demand for this task to each demand of further test intervals. This approximated demand is calculated using the specific utilization of the approximated task on the difference between the considered test interval and the previous test interval.

The test runs on this approximation level until either all relevant test intervals have been checked, a maximum test interval is reached or the approximated demand bound function exceeds the available capacity for a test interval Δt_{fail}. In the last case, the approximation level is raised step-by-step to a higher approximation level ε_{next}. Only those approximation levels are of interest in which at least one of the approximated tasks of the previous level is not approximated any more. For each approximation level a new approximated demand bound function is calculated with exact values for those tasks that are no longer approximated. When the new approximated demand bound function does not exceed the available computation time the test interval is finished and the analysis is continued with the new level. Otherwise the approximation level is raised further. If there are no more approximated tasks at a level, the task set is not schedulable. For each task that is no longer approximated on the new approximation level, the smallest test interval being

Algorithm 6 Dynamic Error Schedulability Test

```
Algorithm DYNAMIC_ERROR
Given: task set Γ
IF U_Γ = Σ_{∀τ∈Γ} c_τ^+ / p_τ > 1 ⇒ not schedulable
k := 1
Δt_max := maximum test interval
testlist := {}
approxlist := {}
Δt_old := 0
∀τ ∈ Γ: ADD te = (NextInt(0, τ), τ) TO testlist
WHILE (testlist ≠ {} and Δt ≤ Δt_max)
   te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
   Δt =INTERVAL OF te
   τ =TASK BELONGING TO te
   REMOVE te FROM testlist
   δ' := δ' + c_τ^+ + (Δt − Δt_old)U_ready
   WHILE (δ' > Δt)
      IF (approxlist = {})
         ⇒ not schedulable
      k := 2k
/** It is possible to use another
/** strategy here for rising the level
      FOR ALL τ' in approxlist with Δt ≤ d_τ' + p_τ'k
         δ' := δ' − ((Δt + j_τ' − d_τ') / p_τ' − ⌊(Δt + j_τ' − d_τ') / p_τ'⌋) c_τ'^+
         ADD te := (NextInt(Δt, τ'), τ') TO testlist
         REMOVE τ' FROM approxlist
         U_r := U_r − c_τ'^+ / p_τ'
      END FOR
   END WHILE
   IF (Δt = d_τ)
      ADD te := (Δt + p_τ − j_τ, τ) TO testlist
   ELSE IF (Δt < d_τ + p_τ k)
      ADD te := (Δt + p_τ, τ) TO testlist
   ELSE
      U_r := U_r + c_τ^+ / p_τ
      τ ADD TO approxlist
   END IF
   Δt_old := Δt
END WHILE
⇒ schedulable
```

larger than Δt_{fail} has to be added to *testlist* (but only if it does not exceed the global maximal test interval). The algorithm is continued until either *testlist* is empty or the analysis fails for a considered test interval.

The event stream version of this test is given in algorithm 7.

4.2. All-approximated algorithm

Once the dynamic error schedulability test has switched to a higher approximation level it remains there. This is especially a problem for cases in which the demand bound

Algorithm 7 Dynamic Error Schedulability Test - Event Stream Version

```
Algorithm DYNAMIC_ERROR_EVENT_STREAM
Given: task set Γ
IF U_Γ = Σ_{∀τ∈Γ} Σ_{∀θ∈Θ_τ} c_τ^+ / p_θ > 1 ⇒ not scheduable
k := 1
Δt_max := maximum test interval of Γ
testlist := {}
approxlist := {}
Δt_old := 0
∀τ ∈ Γ, ∀θ ∈ Θ_τ : ADD te = (NextInt(0, θ), θ) TO testlist
WHILE (testlist ≠ {} and Δt ≤ Δt_max)
   te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
   Δt =INTERVAL OF te
   θ= EVENT ELEMENT BELONGING TO te
   τ = TASK BELONGING TO θ
   REMOVE te FROM testlist
   δ' := δ' + c_τ^+ + (Δt − Δt_old) U_ready
   WHILE (δ' > Δt)
      IF (approxlist = {})
         ⇒ not scheduable
      k := 2k
/** It is possible to use another
/** strategy here for rising the level
      FOR ALL θ' in approxlist with Δt ≤ d_τ' + a + p_θ' k
         U_ready := U_ready − c_τ'^+ / p_θ'
         δ' := δ' − ( (Δt − d_τ' − a_θ') / p_θ' − ⌊ (Δt − d_τ' − a_θ') / p_θ' ⌋ ) c_τ'^+
         ADD te := (NextInt(Δt, θ'), θ') TO testlist
         REMOVE τ' FROM approxlist
         U_r := U_r − c_τ'^+ / p_θ'
      END FOR
   END WHILE
   IF (Δt < d_τ + a_θ + p_θ k)
      ADD te := (Δt + p_θ, θ) TO testlist
   ELSE
      U_r := U_r + c_τ^+ / p_θ
      θ ADD TO approxlist
   END IF
   Δt_old := Δt
END WHILE
⇒ scheduable
```

function do nearly exceed the available capacity for some test intervals and for which therefore it is not allowed to approximate any, even the smallest task, for these test intervals. In these cases it is necessary to raise the approximation level to a highest value required for any interval of the demand bound function. The approximation level has to cover the number of test intervals necessary for the smallest task. In the further execution of the analysis this level is used for all tasks. This could require to test a large number of test intervals. Often only few test intervals are critical and nearly exceed the available execution time.

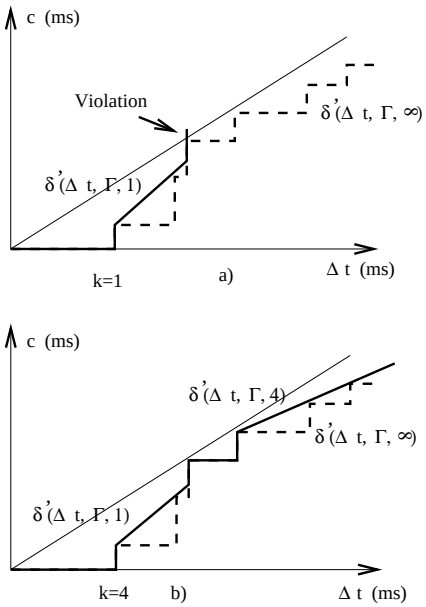

FIGURE 4.2.1. Graphical visualization of the all-approximation algorithm

Only for them the highest approximation level is required. The other test intervals could be approximated using a small approximation level. These considerations lead to a new approach for the analysis.

The idea behind the all-approximation algorithm is to assign to each task an approximation level individually and to raise this level only as far as necessary. Instead of using fixed levels for the complete task set, approximation is done as much as possible. In the algorithm each task is approximated after each of its test intervals. Figure 4.2.1 shows this for an example. Until the first violation the algorithm works in the same way as the dynamic error algorithm. After this violation the approximation restarts for each task as soon as possible again, this means at the next job of each task. Test intervals are only generated and considered at the beginning and when violations have occurred previously.

Therefore no new test interval is inserted into *testlist* after the analysis of any test interval. In the algorithm new test intervals are added to *testlist* only in those cases in which the calculated approximated demand of a test interval exceeds the available computation time.

The algorithm 8 implements these considerations. At the start of the algorithm the first test interval of each task is inserted into *testlist*. The test intervals in *testlist* are processed in ascending order. All the following test intervals of the tasks are approximated first. Therefore each task is added to *approxlist* (and their specific utilization to U_r) after analyzing its test interval. Whenever the approximated demand bound function δ' exceeds the available execution time for a test interval Δt, a step-by-step revision of the approximations of the single tasks is necessary. Only if an approximation of a task is revised in this step, it is necessary to insert a new test interval for this task. It is necessary to insert the first test interval of the task that is larger than Δt. Lemma 4.1.3 gives its calculation. Note

Algorithm 8 All Approximated Test

```
Algorithm All-Approximated
Given: task set Γ
IF U_Γ = Σ_{∀τ∈Γ} c_τ^+/p_τ > 1 ⇒ not schedulable
testlist := {}
approxlist := {}
Δt_old := 0
∀τ ∈ Γ: ADD te = (NextInt(0,τ),τ) TO testlist
WHILE (testlist ≠ {})
   te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
   Δt =INTERVAL OF te
   τ =TASK BELONGING TO te
   REMOVE te FROM testlist
   δ' := δ' + c_τ^+ + (Δt − Δt_old)U_ready
   WHILE (δ' > Δt_act)
      IF (approxlist = {})
         ⇒ not schedulable
      τ' = TASK WITH LARGEST (p_τ − d_τ) OF approxlist
     /** Other orders of
     /** revised elements possible
      δ' := δ' − ((Δt+j_τ'−d_τ')/p_τ' − ⌊(Δt+j_τ'−d_τ')/p_τ'⌋)c_τ'^+
      ADD te := (NextInt(Δt,τ'),τ') TO testlist
      REMOVE τ' FROM approxlist
      U_r := U_r − c_τ'^+/p_τ'
   END WHILE
   U_r := U_r + c_τ^+/p_τ
   ADD τ TO approxlist
   Δt_old := Δt
END WHILE
⇒ schedulable
```

that all skipped intervals smaller than Δt are covered by the previous tested intervals, so it is only necessary to consider intervals larger than Δt. The revision is done by replacing the approximated execution time of a task by its exact execution time using lemma 4.1.1. The revision is done task by task, testing for each task whether the new approximated demand is now covered by the available computation time or not. Only if the execution time still exceeds the capacity at the time when the approximation for all tasks is revised, the system is not schedulable. A task system is recognized as schedulable if there is a test interval Δt in which all tasks can be approximated successfully and therefore the *testlist* is empty. Essential for the number of test intervals is, in which order the revision of the task is done. The best would be to choose that set of tasks for revision that leads to the largest following test interval. Always choosing this set would lead to the largest differences between consecutive test intervals and therefore to the smallest possible number of test intervals. Unfortunately the construction of the set that leads to these largest test intervals is not simple. It can be necessary to consider the test intervals and the exact approximation errors of all other tasks. Even worse, a task with a large following test interval will usually lead only to a small approximation error, whereas a task with a test interval close by can have a

Algorithm 9 All Approximated Test - Event Stream Version

```
Algorithm All-Approximated-Event-Stream
Given: task set Γ
IF U_Γ = Σ_{∀τ∈Γ} Σ_{∀θ∈Θ} c_τ^+ / p_θ > 1 ⇒ not schedulable
testlist := {}
approxlist := {}
Δt_old := 0
∀τ ∈ Γ, ∀θ ∈ Θ_τ : ADD te = (NextInt(0, θ), θ) TO testlist
WHILE (testlist ≠ {})
   te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
   Δt =INTERVAL OF te
   θ = EVENT ELEMENT OF te
   τ =TASK BELONGING TO θ
   REMOVE te FROM testlist
   δ' := δ' + c_τ^+ + (Δt − Δt_old)U_ready
   WHILE (δ' > Δt_act)
      IF (approxlist = {})
         ⇒ not schedulable
      θ' = EVENT ELEMENT WITH LARGEST p_θ −
d_τθ OF approxlist
      τ' = TASK BELONGING TO θ'
     /** Other orders of
     /** revised elements possible
      δ' := δ' − ((Δt − a_θ' − d_τ') / p_θ' − ⌊(Δt − a_θ' − d_τ') / p_θ'⌋) c_τ'^+
      ADD te := (NextInt(Δt, θ'), θ') TO testlist
      REMOVE θ' FROM approxlist
      U_r := U_r − c_τ'^+ / p_θ'
   END WHILE
   U_r := U_r + c_τ^+ / p_θ
   ADD θ TO approxlist
   Δt_old := Δt
END WHILE
⇒ schedulable
```

large concrete approximation error. This is the case if the periods of the two tasks are in the same range. A large distance to the following test interval of a task means a small distance to the previous test interval. This means a small concrete approximation error because it directly depends on the difference between the actual test interval Δt and the previous test interval of the task.

As the order in which the approximation of the tasks should be revised we propose the sizes of the difference between period and deadline of the task. Tasks with large periods have the potential of a large approximation error and will in average lead to large following test intervals. These tasks are therefore most suitable for revising the approximation. In chapter 6 we have compared the run-time of the analysis using different orders for the task for revision.

The event stream version is given in algorithm 9.

4.3. Generalization of the maximum test interval

For the all-approximation analysis it is interesting to consider the test interval after which no violation can happen any more. For a schedulable task set this test interval is not larger than the maximum test intervals proposed by Baruah et al. [19] and by Ripoll et al. [119] (lemma 2.2.13).

LEMMA 4.3.1. *The largest test interval being analyzed by the all-approximation analysis is equal or smaller than the test bound given by Ripoll et al. [119] (lemma 2.2.13)*

To prove this, we need:

LEMMA 4.3.2. *The all-approximation analysis finish its execution at the following test interval* $(\forall \tau \in \Gamma : \Delta t_a > d_\tau)$*:*

$$\Delta t_a = \sum_{\forall \tau \in \Gamma} \left(\frac{\Delta t_a + p_\tau - d_\tau}{p_\tau} \right) c_\tau$$

This lemma follows directly out of the definitions. With this lemma it is easy to prove the previous lemma.

PROOF. We can do the following calculations (condition $d_\tau < p_\tau$):

$$\Delta t_a = \Delta t_a \sum_{\forall \tau \in \Gamma} \frac{c_\tau}{p_\tau} + \sum_{\forall \tau \in \Gamma} \left(\frac{p_\tau - d_\tau}{p_\tau} \right) c_\tau$$

$$\Delta t_a (1 - U_\Gamma) = \sum_{\forall \tau \in \Gamma} \left(\frac{p_\tau - d_\tau}{p_\tau} \right) c_\tau$$

$$\Delta t_a = \frac{\sum_{\forall \tau \in \Gamma} \left(1 - \frac{d_\tau}{p_\tau} \right) c_\tau}{1 - U_\Gamma}$$

The test bound given by Ripoll et al. [119] is the same as the bound resulting of the all-approximation algorithm in the case that all tasks in the task set have a deadline being equal or smaller than their period. If the task set contains tasks having a deadline larger than their period, the bound of the all-approximation algorithm is tighter than the bound given by Ripoll et al. [119]. □

Only the busy period condition can result in a smaller upper bound for the test intervals. But the effort required to calculate the busy period has an exponential complexity and can therefore become quite larger than the whole all-approximation test.

4.4. Complexity

The resulting complexity of the all-approximation test is unknown. Considering one test-interval has a complexity of $O(n \log n)$ with n being the number of tasks. In the worst case the effort for one test interval consists of undoing the approximation of all tasks except the actual task and to insert for each of it the next larger test interval into the sorted test list. The problem for complexity analysis is that the number of test intervals needed by the all-approximation algorithm in the worst-case is still unknown. An upper bound for this number of test intervals is the number needed by the original processor demand

criterion. This original processor demand criterion has a pseudo-polynomial complexity. The number of test intervals depends on the number of tasks in the task set as well as on their parameters. Again the ratio between the smallest and the largest tasks in the task set plays an important role for the number of test intervals.

No tighter complexity bound than pseudo-polynomial is known for the all-approximated algorithm so far. We have used experiments with randomly generated task sets to find a lower bound on the complexity of the all-approximated analysis. We have found out that the new algorithm needs a lot less effort than all previously known exact and approximated analyses. It also seems that its effort is independent of the ratio of the parameters of the tasks and only depends on the number of tasks in a task set and the utilization of the task set. See the evaluations given in chapter 6 for the results.

CHAPTER 5

Approximation for static priority scheduling

Although the EDF scheduling scheme is proved to be optimal and therefore it allows a high utilization of the processing elements, many applications and methods are based on scheduling with static priorities. On the one side, the implementation for this scheduling scheme is said to be simple and it is not necessary to assign deadlines to the tasks. On the other side, static priorities do not allow high utilizations as achievable with EDF. An interesting comparison of the two scheduling schemes can be found in [35]. This paper contradicts many prejudices against the EDF scheduling scheme. Despite that, it is still interesting to find efficient approximative and exact tests for static priorities. In this chapter we want to extend our approximative schedulability test to static priority systems. We propose an approximative test and also an efficient exact test by extending the approximation.

Based on the idea of the superposition approach presented in [5] Fisher and Baruah have proposed an approximative schedulability test for static priority systems, too [51, 52, 53] .

In contrary to their approaches we propose a test based on the same elements we used for EDF scheduling, the sub-additive functions. These functions can be regarded as an abstraction layer between the event models and the test algorithm. Therefore the new test algorithms are suitable for powerful event models, not only for the periodic task model. For example they are suitable for the event stream model. They also allow to easily combine the tests for both scheduling schemes to an integrated powerful overall-test.

5.1. Exact schedulability analysis

Let us first consider some already existing schedulability tests for the preemptive scheduling scheme with static priorities. Each task in a task set has one priority. A task with a higher priority will be preferred over a task with a lower priority. In a preemptive scheduling scheme the execution of a task can be interrupted by a task with a higher priority and be postponed until the execution of this higher-priority task has finished. For the purpose of the following considerations and without loss of generality we will order the tasks with increasing priorities. We demand that each task has a unique priority. Therefore we assume, again without loss of generality, that task τ_1 has a higher priority than τ_2 having a higher priority than τ_3 and so on.

DEFINITION 5.1.1. *Let $prio(\tau)$ be a function giving the priority of task τ. Let $hp(\tau,\Gamma)$ be the task set containing all those tasks of Γ having a priority higher than τ*

$$hp(\tau,\Gamma) = \{\tau_x \mid prio(\tau_x) > prio(\tau)\}$$

For static priority schedulability analysis it is not sufficient to consider only the demand bound function, which covers the amount of workload finished within the given interval. For EDF it is sufficient to consider the demand bound function, as only the workload covered by this function has priority and is therefore preferred to all remaining workload. For static priority scheduling, workload that has to be finished later than other workload can originate from tasks having a higher priority then the tasks responsible for the other workload and is then preferred over this other workload. The total amount of the preferred workload can be captured by the request bound function.

DEFINITION 5.1.2. *Request Bound Function*

The request bound function $\rho(\Delta t, \Gamma)$ returns for each interval-length Δt the cumulated worst-case workload of all jobs of all task getting ready for execution within the interval Δt.

$$\rho(\Delta t, \Gamma) = \sum_{\forall \tau \in \Gamma} \left\lceil \frac{\Delta t}{p_\tau} \right\rceil c_\tau$$

This request consists of the execution time of those jobs for which the invocation time is within the interval I. For a single task we will write $\rho(\Delta t, \tau)$. For an interval Δt and a task τ a chain of consecutive jobs of the task contributes fully to the request bound function $\rho(\Delta t, \tau)$ if the difference between the release time of the first job of the chain and the release time of the last job is equal or smaller than the length of Δt.

The execution of a job of a task is postponed by the request of all tasks having a higher priority which is given by the request bound function for these tasks ($\rho(\Delta t, hp(\tau, \Gamma))$. Therefore it is necessary to consider the feasibility test for each task separately.

We will first consider the case that the possible execution intervals for the jobs of task τ cannot overlap.

DEFINITION 5.1.3. *(Task separation constraint) A task τ fullfills the task separation condition if the possible execution interval (e.g. the interval including every point in time in which the job can execute) of each job of τ does not overlap with the execution intervals of the previous and of the next following job of the same task. This is the case if the absolute deadline of each job of the task occurs earlier than the release of the next following job.*

The task separation constraint allows simplifying feasibility analysis. We will introduce and prove a schedulability analysis with the following definitions and lemmas

DEFINITION 5.1.4. *(Satisfaction interval - schedule formulation) Let job $\tau_{i,j}$ belong to a task τ_i bound on a resource ρ holding the task separation condition. Each interval Δt for which the following condition holds is called a satisfaction interval for τ:*

- *The interval starts at an idle point with respect to the task and all tasks with higher priorities. That means that exactly at the start point of the interval only those jobs are ready for execution that have just arrived at the start point. Tasks with a lower priority than τ are not considered.*
- *The job $\tau_{i,j}$ becomes ready exactly at the start of the interval*

- *The capacity for Δt does meet or exceed the sum of the request bound function for Δt with respect to the higher priority task set of τ bound on one resource ρ and the execution time for the job itself.*

$$\chi(\Delta t, \rho) \geq \rho(\Delta t, hp(\tau, \Gamma)) + c_\tau^+$$

This satisfaction interval is based on one concrete (worst-case) schedule only. In the following lemmas we will show how the satisfaction interval is used for schedulability analysis. The separation condition allows reducing the complete schedulability analysis to a check of the satisfaction intervals for every first job of each task.

In those cases in which the separation constraint does not apply, it is not longer possible to check only the first job of each task. Instead it is necessary to analyze every job within the first busy period of each task. In definition 5.1.7 we will propose a modified functional description of a satisfaction interval for which the separation condition is no longer necessary. But let us first continue with the schedulability analysis and the proof of it.

LEMMA 5.1.5. *A job is always executed completely within its satisfaction interval.*

PROOF. Let us assume there is a job of a task τ and an interval Δt fulfilling the condition of the satisfaction intervals of definition 5.1.4. The job is not executed fully, despite that the capacity meets or exceeds the execution time of the job and the request bound function for the higher priority tasks. By definition of the satisfaction interval, the job gets ready at the start of the interval and remains ready during the complete length of Δt. Therefore the processor is not idle at any point of time within Δt as the job would execute at this point of time. Also tasks with lower priorities cannot be executed within Δt, as τ would execute instead of them. No job that has arrived before the start of Δt can execute in Δt as this would be in contradiction to the idle point condition for the start point of the satisfaction interval. So, as the capacity is fully used by the job and the tasks with is higher priority than τ either the execution time of the job would exceed c_τ^+ or the sum of the execution time of the tasks with is higher priority then τ would exceed $\rho(\Delta t, hp(\tau, \Gamma))$. Both are in contradiction to the definitions of the two values. Therefore the assumption does not hold. □

With this lemma we can now formulate and prove the schedulability analysis itself.

THEOREM 5.1.6. *A task τ holding the task separation condition is schedulable (finishes always its execution before its deadline) if there exists a satisfaction interval $\Delta t \leq d_\tau$ for the simultaneous release of the first job of τ and a job of each task having a higher priority than τ.*

$$\forall \tau \in \Gamma : \exists \Delta t \leq d_\tau \,|\, \chi(\Delta t, \rho) \geq c_\tau^+ + \rho(\Delta t, hp(\tau, \Gamma))$$

PROOF. Let us first prove, that the simultaneously release of all tasks of a task set is the worst-case situation for the task with the lowest priority within the task set. The worst-case situation for a job occurs if it's processing achieves the longest delay. Consider the top two tasks τ_1, τ_2 with the highest priorities and assume that they are not released

simultaneously. The release of a job of τ_2 can occur either during the processing of a job of τ_1 or during an idle time of the processor. In the first case the release time of the job of τ_2 can be moved to the release time of the running job of τ_1 without changing the concrete schedule. In the second case the release time of the job can be moved to the release time of the next job of τ_1. The amount of workload that was originally processed between the original release time and the new release time of the job will, in the modified schedule, have to be also moved. It will fill the idle time following the new simultaneous release time in a size that is equal to the difference between the original and the new release time. Therefore the shift operation to a simultaneous release of tasks cannot lead to a shorter delay for one of the tasks. The same proof can now be used recursively on all following tasks. Therefore the simultaneous release of all tasks is the worst-case situation.

As we know by lemma 5.1.5 the execution of a job is completely finished within each of its satisfaction intervals in those cases in which the task separation condition holds. If there exists a satisfaction interval that does not exceed the relative deadline and the task separation condition holds, each job finishes within the relative deadline of the task and the task is schedulable. □

The problem is that the possible satisfaction intervals are somewhere between the worst-case execution time of the task and its relative deadline. Neither of these values needs to be such a satisfaction interval. The satisfaction interval is similar to the definition of a scheduling point in [80]. It can be time consuming to find one satisfaction interval. The question is, if there is a more efficient way to prove schedulability than calculating the worst-case response time.

For the general deadline case, in which the task separation condition does not hold, it is necessary to use an extended definition of satisfaction interval and to find these intervals for several jobs of the same task. The satisfaction intervals still have to start at an idle point but can include several jobs of the considered task.

DEFINITION 5.1.7. *(Satisfaction interval - functional formulation) Let us consider each task τ separately. An interval Δt' is a satisfaction interval for a task τ and an interval Δt if $\Delta t' \leq \Delta t$ and the sum of the demand of the task for Δt and the request bound function for $\Delta t'$ of all tasks with a higher priority then τ does not exceed the capacity with respect to $\Delta t'$:*

$$\chi(\Delta t', \rho) \geq \delta(\Delta t, \tau) + \rho(\Delta t', hp(\tau, \Gamma))$$

The satisfaction intervals are visualized in figure 5.1.1. In it the function $F(\Delta t') = \delta(\Delta t, \tau) + \rho(\Delta t', hp(\tau, \Gamma))$ is plotted together with the capacity function $\chi(\Delta t', \rho)$. There are three spaces of satisfaction intervals.

This is the general description for a satisfaction interval. For a complete real-time analysis it is necessary to find such a satisfaction interval $\Delta t'$ for all jobs of each task. Let $\Delta t_{\tau,k} = (k-1)p_\tau + d_\tau$ be the interval of the k-th job of τ.

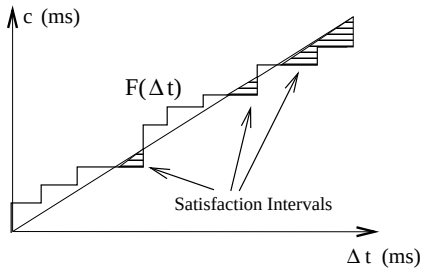

FIGURE 5.1.1. Example of satisfaction intervals

LEMMA 5.1.8. *A task set is feasible if for each task and each of its intervals $\Delta t_{\tau,i}$ such a satisfaction interval exists.*

$$\forall \Delta t | \delta(\Delta t, \tau) > 0 : \ \exists \Delta t' \leq \Delta t | \chi(\Delta t', \rho) \geq \delta(\Delta t, \tau) + \rho(\Delta t', hp(\tau, \Gamma))$$

PROOF. Let, without loss of generality, $\Delta t'$ be an satisfaction interval belonging to a task τ and an interval $\Delta t_{\tau,i}$ with $\Delta t' \leq \Delta t_{\tau,i}$. The unified starting point of both $\Delta t_{\tau,i}$ and $\Delta t'$ is set to the last idle point of their processor before the end point of $\Delta t'$. Assume that the job τ_i of task τ_i having its absolute deadline given by the end of $\Delta t_{\tau,i}$ misses its deadline despite that the condition of the satisfaction interval $\Delta t'$ holds. By definition the complete costs of the job are included in the value of the demand bound function $\delta(\Delta t, \tau)$. For missing the deadline it is necessary that at least one part of the demanded workload cannot be processed within Δt. The sum of the demand bound function and the request bound function for $\Delta t'$ does not exceed the minimal available supply of computational capacity. As one part of the demand bound function is not processed within $\Delta t'$ either the processor is idle somewhere within $\Delta t'$ or it processes tasks that are not covered by the demand and the request bound function. The processor cannot be idle as otherwise the job that misses its deadline would be processed at the idle time. Also no task with a lower priority can execute within $\Delta t'$ for the same reason. No job occurring before the start of $\Delta t'$ can execute within $\Delta t'$ as the start point of $\Delta t'$ is an idle point of the processor. By the definition of an idle point there is no processable job at this point of time having arrived before this point of time. All tasks with higher priorities arriving within $\Delta t'$ are included in the request bound function and therefore covered by the service function. The jobs of the task itself have all the same relative deadline. All jobs having an arrival time before the arrival time of τ also have a deadline before the deadline of τ. Those jobs of the task having arrived before the start of $\Delta t'$ are completely executed before the start of $\Delta t'$ due to the idle point condition. The computational effort of all those jobs of the same task having arrived within $\Delta t'$ but before τ and can therefore delay the execution of τ is covered by the demand bound function $\delta(\Delta t, \tau)$ and therefore also by the service bound function. It is not possible to find a schedule in which τ misses its deadline. □

This results in a huge effort for the analysis. The concept of the satisfaction intervals can also be used to determine the worst-case response time of a task. Again we consider the

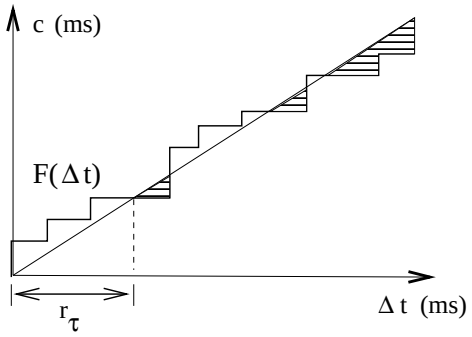

FIGURE 5.1.2. Worst-case response-time with satisfaction intervals

cases in which the task separation condition holds or not separately. In case the condition holds we only have to consider the first job of the task.

THEOREM 5.1.9. *The worst-case response time of a task r_τ for which the separation condition (definition 5.1.3) holds is given by the smallest satisfaction interval as defined in definition 5.1.7.*

$$r(\tau) = min(\Delta t | \rho(\Delta t, hp(\tau,\Gamma)) + c_\tau^+ \leq \chi(\Delta t, \rho))$$

PROOF. As the satisfaction intervals are sufficient to prove schedulability, the worst-case response time cannot be larger than this interval. So the remaining question is, if it can be smaller than the first satisfaction interval. The job can only be executed if no job with a higher priority is ready to execute and have a remaining execution demand. This is the case for those intervals in which the execution of all higher priority jobs can be completely satisfied by the service bound function. To complete the execution of the job itself it is additionally necessary that the service function also provide enough capacity for this job and all previous jobs of the task. This is the case at the first satisfaction interval. It is therefore the worst-case response time of the task. □

The theorem is visualized in figure 5.1.2. The worst-case response-time is the first satisfaction interval.

In case that the separation condition does not hold, the worst-case response time can be the response time of any job within the first busy period. In this case it is necessary to calculate it for every one of these jobs.

LEMMA 5.1.10. *(similar to [130]) The worst-case response time for a specific job (the first, second, third,....) is given by the difference between the smallest interval in which the job can arrive (given by the interval bound function $\psi(i,\tau)$) and the next following satisfaction interval $(\rho(\Delta t, hp(\tau,\Gamma)) + c_\tau^+ i \leq \chi(\Delta t, \rho))$. The overall worst-case response time is given by the maximum of all worst-case response times of the jobs.*

$$r_\tau = max_{\forall i \in \mathbb{N}}(r_{\tau,i} - \psi(i,\tau))$$
$$r_{\tau,i} = min(\Delta t | \rho(\Delta t, hp(\tau,\Gamma)) + c_\tau^+ i \leq \chi(\Delta t, \rho))$$

We can also reformulate this calculation so that it is independent of the jobs and is only based on intervals.

THEOREM 5.1.11. *The overall worst-case response time $r(\tau)$ for a task τ is given by:*

$$r(\tau) = max(\Delta t | r(\Delta t, \tau) - \Delta t)$$
$$r(\Delta t, \tau) = min(\Delta t' | \eta(\Delta t, \tau) r_\tau + \rho(\Delta t', hp(\tau, \Gamma)) \leq \chi(\Delta t', \rho))$$

PROOF. Let us assume that there exists a job τ_i with a longer response time than given by the equations above. By definition of the fixed-priority scheduling the resource ρ has to be completely busy between the release of τ_i and the finishing time of τ_i. Let $\Delta t''$ be the interval between the finishing time of τ_i and the last previous idle time. Let us assume a synchronous release of jobs of all tasks at the start of $\Delta t''$. All other release pattern would only lead to equal or less costs within Δt and therefore to an equal or smaller $\Delta t''$. Let us also, without loss of generality, denote the job of τ released at the begin of $\Delta t''$ with τ_1. Due to the assumption there exists an interval $\Delta t'$ fulfilling $\eta(\psi(i,\tau),\tau) r_\tau + \rho(\Delta t', hp(\tau,\Gamma)) \leq \chi(\Delta t', \rho)$ with $\Delta t' < \Delta t''$. As $\Delta t'$ is an idle point with regard to τ and $hp(\tau)$ the job τ_i has finished at $\Delta t'$ which is in contradiction to the assumption. □

To find the worst case it is necessary to analyze all jobs within the busy period of the tasks and all higher priority tasks.

5.2. Exceeding costs

The disadvantage of all the approaches proposed in the last section is that it is necessary for proving the schedulability of one interval Δt to find or prove the existence of an additional interval $\Delta t'$ with unknown size. For integrating the analysis in an analysis framework it would be better if it would be possible to use only one interval Δt for all functions. To find a schedulability analysis satisfying this constraint an additional function is required, the exceeding-cost function.

DEFINITION 5.2.1. *(Exceeding Cost Function) $\mathscr{E}(\Delta t, \Gamma)$*

The exceeding costs are those part of the execution times of all jobs of all tasks in the task set, arriving within Δt that cannot be processed within Δt independently of the used scheduling scheme of the processor.

The exceeding costs are those costs of the different jobs that cannot be processed within the test interval due to the late arrival time of the jobs.

EXAMPLE 5.2.2. Consider the example given in figure 5.2.1. It is the task set $\Gamma = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ with $\tau = (p_\tau, c_\tau^+, d_\tau)$ and $\tau_1 = (8\,ms, 4\,ms, 4\,ms)$, $\tau_2 = (22\,ms, 3\,ms, 7\,ms)$, $\tau_3 = (19\,ms, 3\,ms, 17\,ms)$, $\tau_4 = (30\,ms, 1\,ms, 26\,ms)$. Let us consider the exceeding costs for the first job $\tau_{4,1}$ of task τ_4. Let $d_{\tau_4} = 50$ ms therefore $\Delta t = 50$ ms is the considered interval for the exceeding costs. The job $\tau_{1,4}$ of task τ_1 arrives at time 48 ms (or more exactly not before the end of an interval with length 48) having an execution time of 4 ms. Even if this job can use fully the available capacity in the remainder of the interval for execution it is only possible that this job executes for two ms within Δt. So it is not possible for the remaining two ms to be executed within Δt. They belong to the exceeding costs. A set of jobs arriving late can contribute even more to the exceeding costs than the single

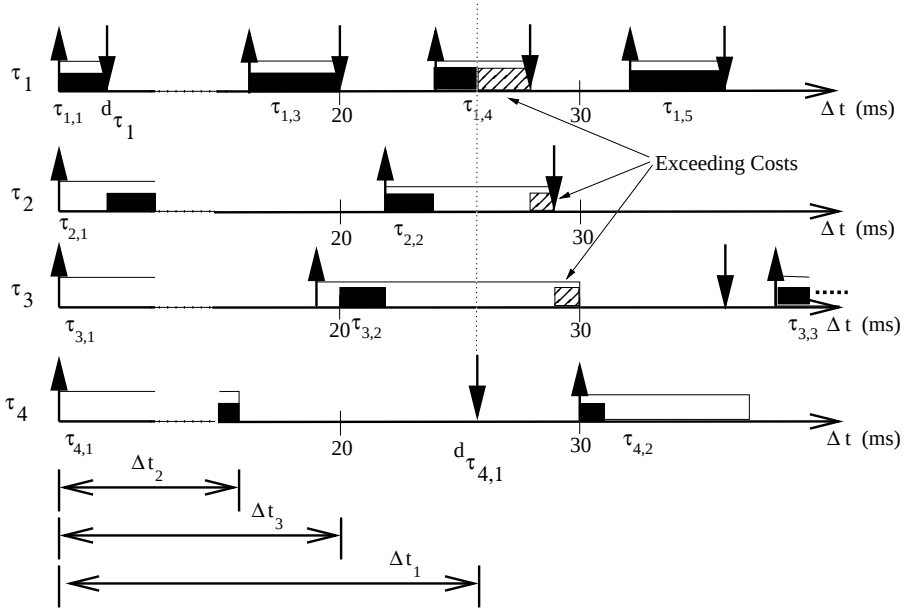

FIGURE 5.2.1. Task set example for exceeding costs

jobs of the set. They cannot execute concurrently. In the example the job $\tau_{2,j}$ arriving at time point 45 ms having an execution time of 5 ms would not lead to exceeding costs when considered alone. Together with job $\tau_{1,i}$ it leads to a value of 4 ms for the exceeding costs, even more than $\tau_{1,i}$ produces alone. Two milliseconds are the exceeding costs of $\tau_{1,i}$ the other two milliseconds occur because $\tau_{2,j}$ cannot be executed fully within Δt due to its postponing by job $\tau_{1,i}$. Note that the scheduling of these two jobs is not relevant here. One of the two jobs is shifted partly out of the boarders of the interval by the other job. For the exceeding cost function it is not important which job is shifted and therefore contributes to the exceeding costs. Only the sum of all computation time that is guaranteed to be not within the interval is of relevance. In the example the job $\tau_{3,k}$ also contributes to the exceeding costs but only when its delay by the job $\tau_{1,i-1}$ is taken into account. This shows that, especially to the concurrency of the jobs of different tasks, several jobs of a task can cause a contribution to the exceeding costs. Each job between the last idle point within an interval and the end of the interval can cause exceeding costs and has therefore to be taken into account for the exact calculation of them.

LEMMA 5.2.3. *The exceeding costs for an interval Δt and a task set Γ are given by*

$$\mathscr{E}(\Delta t) = \rho(\Delta t, \Gamma) - \rho(\Delta t', \Gamma) - (\chi(\Delta t) - \chi(\Delta t'))$$
$$\Delta t' = max(\Delta t'' | \Delta t'' \leq \Delta t \wedge \rho(\Delta t'', \Gamma) \leq \chi(\Delta t''))$$

PROOF. Let Δt_1 be the requested interval Δt, Δt_2 be the interval $\Delta t'$. The intervals are visualized again in figure 5.2.1. A necessary condition for the existence of exceeding costs is that in an interval Δt_3 more computation is requested by Γ than capacity is available. The exceeding costs of interval Δt_3 are those part of the costs requested within Δt_3 that exceeds the capacity available within Δt_3. Relevant for Δt_3 are only those intervals starting

after the last idle point, as no costs occurring before this idle point can contribute to the exceeding costs. So we have $\Delta t_3 \geq \Delta t_1$.

To prove the condition we assume that an interval Δt_3 exists with $\Delta t_3 > \Delta t_2$ leading to more exceeding costs than Δt_2. Any job arriving between the start of Δt_2 and the start of Δt_3 and which is not completely finished processing at the start of interval Δt_3 contributes to the exceeding costs. So a necessary condition for Δt_3 is that no such job exists. But then the end of Δt_3 is an idle point which is in contradiction to the assumption that the end of $\Delta t'$ is the last idle point. □

A new schedulability test can be formulated using the exceeding cost function.

THEOREM 5.2.4. *Let τ be a task of task set Γ. τ always meets its deadline if the sum of the demand bound function of the task and the request bound functions of $hp(\tau,\Gamma)$ is lower or equal to the available execution time in Δt and the exceeding costs of $hp(\tau,\Gamma)$:*

$$\delta(\Delta t,\tau)+\rho(\Delta t,hp(\tau,\Gamma)) \leq \chi(\Delta t)+\mathscr{E}(\Delta t,hp(\tau,\Gamma))$$

PROOF. The exceeding costs cannot by definition be executed within Δt, but they are included in $\delta(\Delta t,\tau)+\rho(\Delta t,hp(\tau,\Gamma))$. Therefore a virtual capacity in size of the exceeding costs is additionally available within Δt to satisfy $\delta(\Delta t,\tau)+\rho(\Delta t,hp(\tau,\Gamma))$. □

The algorithm calculating the exceeding costs exactly is given in algorithm 10. It covers the period task model with jitter. Having no jitter, the jitter can be set to zero and it is only necessary to check whether the first job for every task keeps its deadline.

The idea behind the algorithm is quite simple. The jobs occurring in the interval Δt are considered step-by-step backwards, starting with the last job. We consider a set of last jobs of the interval being increased by one job at each step. The interval in which all these jobs occur is called the remaining interval Δt_r. It is given by the difference of the end of the complete interval and the release time of the first one of the last jobs. We have to consider two kinds of values here. First, there is the execution time of these last jobs. One part of the cumulated execution times of these jobs are handled within the remaining part of the interval, the other part of the cumulated execution times forms the exceeding costs. As already mentioned the exceeding costs cannot be executed within the interval Δt. To calculate the size of these parts we have to consider the maximum amount of execution time that can be handled within the remaining interval. This amount of processable execution time is given by the difference of the capacity of the complete interval Δt and the capacity of an interval $\Delta t' = \Delta t - \Delta t_r$. Note that both capacities deliver the minimum processable execution time, so the difference of these functions can exceed the minimum execution time that is available for any interval of length Δt_r.

This remaining execution time is also calculated step-by-step in the proposed algorithm by considering the service bound function at the release times of the jobs. The exceeding costs are calculated by subtracting this difference of the service bound functions from the cumulated execution times of the considered jobs. The event stream version is given in algorithm 11 .

Algorithm 10 Exceeding-Cost Analysis

```
Algorithm Exceeding-Cost-Analysis
Given: task set Γ, τ
IF U_Γ = Σ_{∀τ∈Γ} c_τ^+ / p_τ > 1 ⇒not schedulable
Δt_bp = BusyPeriod(τ ∪ Γ_hp(τ))
Δt_base := d_τ
while (Δt_base ≤ Δt_bp)
    testlist := {}
    Δt_old := Δt_base
    C_r = δ(Δt_old, τ) + ρ(Δt + j_τ, Γ_hp(τ)) − χ(Δt_old)
    ∀τ' ∈ Γ_hp(τ) : ADD te := ( ( ⌊ (Δt_old + j_τ') / p_τ' ⌋ p_τ' − j_τ' ), τ' ) TO testlist
    WHILE (testlist ≠ {})
        IF (C_r ≤ 0)
            ⇒schedulable
        END IF
        te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
        Δt_act =INTERVAL OF te
        τ =TASK BELONGING TO te
        REMOVE te FROM testlist
        C_r := C_r − c_τ^+ + (χ(Δt_act) − χ(Δt_old))
        IF (Δt_act > 0)
            ADD te := (max(0, Δt_act − p_τ), τ) TO testlist
        END IF
        Δt_old := Δt_act
    END WHILE
    IF (Δt_base = d_τ)
        Δt_base := Δt_base + p_τ − j_τ
    ELSE
        Δt_base := Δt_base + p_τ
    END IF
END WHILE
⇒ not schedulable
```

Note, that it is only necessary to proceed with the calculation of the exceeding cost function until it covers completely the difference between the request bound function of the higher priority tasks and the demand bound function of the task in question on the one side and the service function on the other side. Then it is proven that there is enough processable execution time available within Δt for the job in question to keep its deadline. In the case that the sum of the request and the demand bound function does not exceed the service bound function for the interval Δt no calculation of the exceeding costs is needed at all. The advantage of this approach is that only in case of a very high utilization of the processor it is necessary to calculate the exceeding cost function accurately requiring an effort comparable to the effort for the previous existing tests.

It is possible to re-define the existing schedulability analysis using sub-additive demand, request and exceeding cost function. In doing this the relationships between the different approaches will become more obviously.

Algorithm 11 Exceeding-Cost Analysis - Event stream Version

```
Algorithm Exceeding-Cost-Analysis
Given: task set Γ, τ, θ
IF U_Γ = Σ_{∀τ∈Γ} Σ_{∀θ∈Θ_τ} c_τ^+ / p_τ > 1 ⇒ not schedulable
Δt_bp = BusyPeriod(τ ∪ Γ_hp(τ))
Δt_base := d_τ + a_θ
while (Δt_base ≤ Δt_bp)
    testlist := {}
    Δt_old := Δt_base
    C_r = δ(Δt_old, τ) + ρ(Δt, Γ_hp(τ)) − χ(Δt_old)
    ∀τ' ∈ Γ, ∀θ' ∈ Θ_τ : ADD te :=
((⌊(Δt_old − a_θ') / p_θ'⌋ p_θ' + a_θ'), θ) TO testlist
    WHILE (testlist ≠ {})
        IF (C_r ≤ 0)
            ⇒schedulable
        END IF
        te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
        Δt_act =INTERVAL OF te
        θ' = EVENT ELEMENT OF te
        τ =TASK BELONGING TO θ'
        REMOVE te FROM testlist
        C_r := C_r − c_τ^+ + (χ(Δt_act) − χ(Δt_old))
        IF (Δt_act > 0)
            ADD te := (Δt_act − p_θ'), τ) TO testlist
        END IF
        Δt_old := Δt_act
    END WHILE
    Δt_base := Δt_base + p_θ
END WHILE
⇒ not schedulable
```

5.3. Approximation of Static Priorities

For an approximative analysis for static priority scheduling we have to distinguish between the case of non-arbitrary deadlines and arbitrary deadlines. In the non-arbitrary deadline case only tasks with a deadline smaller than their period are allowed. A schedulable task set with non-arbitrary deadline fulfills the separation condition of definition 5.1.3 and to prove schedulability it is only necessary to check for task sets with non-arbitrary deadlines the first job of each task within a synchronous release of all tasks of the task set. In the case of task sets with arbitrary deadlines, tasks are allowed having a deadline larger than their period, so the possible execution window of several jobs of a task can overlap. To prove schedulability for task sets with arbitrary deadlines it is necessary to prove several jobs of each task. In the worst case it is necessary to prove the schedulability of all jobs within the busy period of the task set.

5.3.1. Non-arbitrary deadline case. Fisher and Baruah [51], [52] have exchanged the request bound function with an approximated request bound function following the

same ideas as already introduced for the approximated demand bound function. The specific approximated request bound function for a task is equal to the specific exact request bound function for the same task up to a limited number of jobs of the task which is determined by the approximation error. After these jobs the exact request bound function is approximated by the specific utilization of the task, same as with the approximated demand bound function.

The complete request bound function is given by the sum of all specific request bound functions, so the complete approximated request bound function is also given by the sum of all specific approximated request bound functions.

For feasibility analysis of task sets with non-arbitrary deadlines it is only necessary to exchange in theorem 5.1.6 the request bound function $\rho(\Delta t, \Gamma)$ by the approximated request bound function $\rho(\Delta t, \Gamma, k)$ with k being the number of exactly considered test intervals. It is necessary to find an interval $\Delta t'$ being equal or smaller than the first deadline of the task and in which the sum of the request bound function of all tasks with a higher priority and the execution time of one job of the task does not exceed the available capacity given by the service bound function for the interval $\Delta t'$.

LEMMA 5.3.1. *[51] A task τ in accordance to the task separation condition for which no interval $\Delta t'$ exists with $\Delta t' \leq d_\tau$ and with*

$$c_\tau^+ + \rho(\Delta t', \Gamma_h(\tau, \Gamma), k) \leq \chi(\Delta t')$$

is not schedulable on a processor with a capacity function $\chi'(\Delta t) = (1 - \frac{1}{k})\chi(\Delta t)$.

PROOF. The proof for this lemma is corresponding to the proof for theorem 3.1.4 and can also be found in [51]. □

We will now propose an approximative analysis using the exceeding cost function, shown in algorithm 12.

The algorithm uses the approximative request and demand bound functions. It starts at the deadline of the first job and calculates the exceeding costs step-by-step until $c_\tau^+ + \rho(d_\tau, hp(\tau, \Gamma), k) \leq \chi(d_\tau) + \mathscr{E}(d_\tau, hp(\tau, \Gamma), k)$. The exceeding costs are the part of the sum of the worst-case execution times which exceeds the available capacity. For the calculation of the exceeding costs we have to consider the last jobs occurring before d_τ, one by one. For each job we have to consider its worst-case execution time $c_{\tau'}^+$ and the amount of capacity available within the remaining part of d_τ. The algorithm calculates an initial amount of uncovered costs C_r at d_τ and than adapts this value with each job considered for the exceeding costs. Therefore the last jobs of each task occurring before d_τ are inserted into *testlist*. It is also necessary to register which higher priority tasks are already approximated at d_τ. For them not the last job is inserted into *testlist* but the last job that is considered exactly.

The value C_r is reduced by the worst-case execution time of the job $c_{\tau'}^+$ and the additional available capacity is added $\chi(\Delta t_{old}) - \chi(\Delta t_{act})$. Furthermore, to take care of the approximation, the amount C_r is reduced by an additional lower bound of the costs required at least by the approximated tasks. This value $((\Delta t_{old} - \Delta t_{act})U_r)$ is calculated with

Algorithm 12 Exceeding-Cost Approximation I

Algorithm Exceeding-Cost-Approximation
(non-arbitrary case)
Given: task set $\Gamma_{hp(\tau)}$, τ, k
IF $U_\Gamma = \sum_{\forall \tau \in \Gamma} \frac{c_\tau^+}{p_\tau} > 1 \Rightarrow$ *not schedulable*
$testlist := \{\}$
$approxlist := \{\}$
$U_r := 0$
$\Delta t_{old} := d_\tau$
$C_r = c_\tau^+ + \rho(\Delta t + j_\tau, \Gamma_{hp(\tau)}, k) - \chi(\Delta t_{old})$
FOR ALL $\forall \tau' \in \Gamma_{hp(\tau)}$
  IF $(kp_{\tau'} - j_{\tau'} \leq \Delta t_{old})$
    ADD θ' TO $approxlist$
    ADD $te := (\max(0, (kp_{\tau'} - j_{\tau'})), \theta')$ TO $testlist$
    $U_r := U_r + \frac{c_{\tau'}^+}{p_{\tau'}}$
  ELSE
    ADD $te := \left(\max\left(0, \left(\left\lfloor \frac{\Delta t_{old} + j_{\tau'}}{p_{\tau'}} \right\rfloor p_{\tau'} - j_{\tau'}\right)\right), \theta\right)$ TO $testlist$
  END IF
END FOR
WHILE $(testlist \neq \{\})$
  IF $(C_r \leq 0)$
    $\Rightarrow$schedulable
  END IF
  te=TEST LIST ELEMENT WITH LARGEST Δt IN $testlist$
  Δt_{act} =INTERVAL OF te
  τ' = TASK OF te
  REMOVE te FROM $testlist$
  $C_r := C_r + (\chi(\Delta t_{old}) - \chi(\Delta t_{act})) - (\Delta t_{old} - \Delta t_{act}) U_r$
  IF $(\tau' \in approxlist)$
    REMOVE τ FROM $approxlist$
    $U_r := U_r - \frac{c_{\tau'}^+}{p_{\tau'}}$
  ELSE
    $C_r := C_r - c_{\tau'}^+$
  END IF
  IF $(\Delta t_{act} > 0)$
    ADD $te := (max(0, \Delta t_{act} - p_{\tau'}), \theta)$ TO $test\text{-}list$
  END IF
  $\Delta t_{old} := \Delta t_{act}$
END WHILE
$\Rightarrow$ *not schedulable*

the cumulated specific utilization of all approximated tasks U_r, same as in the algorithm 1 for the approximative analysis of dynamic priority scheduling. It can happen that we reach the starting point of the approximation for one of the approximated tasks. A test point for this task in *testlist* markes this point. So, if a test point is related to a task in the list of approximated tasks, the approximation for this task is withdrawn and the task is handle exactly during the further calculations. A task is schedulable if $C_r \leq 0$ for any considered job, so the algorithm can stop when the condition is reached for the first time.

Algorithm 13 Exceeding-Cost Approximation I - Event Stream Version

Algorithm Exceeding-Cost-Approximation - Event-Stream-Version
(non-arbitrary case)
Given: task set $\Gamma_{hp(\tau)}$, τ, k
IF $U_\Gamma = \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau} \frac{c^+_\tau}{p_\theta} > 1 \Rightarrow$ *not schedulable*
$testlist := \{\}$
$approxlist := \{\}$
$U_r := 0$
$\Delta t_{old} := d_\tau$
$C_r = c^+_\tau + \rho(\Delta t, \Gamma_{hp(\tau)}, k) - \chi(\Delta t_{old})$
FOR ALL $\forall \tau' \in \Gamma_{hp(\tau)} \forall \theta' \in \Theta_\tau$:
  IF $(a_{\theta'} + k p_{\theta'} \leq \Delta t_{old})$
    ADD θ' TO *approxlist*
    ADD $((a_{\theta'} + k p_{\theta'}), \theta')$ TO *testlist*
    $U_r := U_r + \frac{c^+_{\tau'}}{p_{\theta'}}$
  ELSE
    ADD $te := \left((\left\lfloor \frac{\Delta t_{old} - a_{\theta'}}{p_{\theta'}} \right\rfloor p_{\theta'} + a_{\theta'}), \theta \right)$ TO *testlist*
  END IF
END FOR
WHILE $(testlist \neq \{\})$
  IF $(C_r \leq 0)$
    $\Rightarrow$schedulable
  END IF
  te=TEST LIST ELEMENT WITH SMALLEST Δt IN *testlist*
  Δt_{act} =INTERVAL OF *te*
  θ' =EVENT ELEMENT OF *te*
  τ' = TASK BELONGING TO θ'
  REMOVE *te* FROM *testlist*
  $C_r := C_r + (\chi(\Delta t_{old}) - \chi(\Delta t_{act})) - (\Delta t_{old} - \Delta t_{act}) U_r$
  IF $(\theta' \in approxlist)$
    REMOVE θ' FROM *approxlist*
    $U_r := U_r - \frac{c^+_{\tau'}}{p_{\theta'}}$
  ELSE
    $C_r := C_r - c^+_{\tau'}$
  END IF
  IF $(\Delta t_{act} > 0)$
    ADD $te := (\Delta t_{act} - p_{\theta'}, \theta)$ TO *testlist*
  END IF
  $\Delta t_{old} := \Delta t_{act}$
END WHILE
$\Rightarrow$ *not schedulable*

The version for event streams is given in algorithm 13.

This algorithm matches with our approach for EDF scheduling.

5.3.2. Arbitrary case. For the arbitrary cases the algorithm is more complicated as it is necessary to consider more than one job of each task. The problem is, that a job of a task that is not processed completely before the arrival of the next following job of the same task will delay this next following job. As by the release time of the first job no other

job of the task is pending, this job is not affected by any delay due to previous jobs of the same task. The following jobs of the task can receive a delay and therefore may require a longer response time than the first job. Therefore we have to consider in the arbitrary case the response time of more than one job for each task.

The worst-case response time has to happen somewhere within the first busy period of a task set consisting only of the task in question and all tasks with a equal or higher priority. All jobs of the task in question having arrived within this first busy period are also processed completely within this interval. The following jobs will not receive any delay from these jobs.

THEOREM 5.3.2. *Let Γ be a task set and $\tau_i \in \Gamma$ be the task with the lowest priority within Γ. If any job of τ_i fails to meet its deadline also one job of τ_i fails to meet its deadline within the first busy period of Γ.*

PROOF. We will prove the contra-positive. Let τ_f be the job that fails to meet its deadline. Each job belongs to one busy period starting at an idle point of the system. In case that the busy period of τ_f starts with a simultaneous release of all higher-priority tasks it would be equal to the first busy period and therefore one job within the first busy period would fail to meet its deadline. In case that the busy period of τ_f does not start with a synchronous release of all higher-priority tasks we can shift the release times of these tasks until we get a synchronous release. As shown in [88] a shift of release-times to a synchronous release can only lead to more interruptions of lower-priority tasks, therefore only to longer delays. The job τ_f would still fail its deadline and, as the busy period with the synchronous release is equal to the first busy period, a job within the first busy period would also fail its deadline. □

For proving the schedulability for τ we have to check every job of τ within its first busy period. Fisher and Baruah [52] propose a complicated solution of this problem, but using the approximated request bound function a bound for the maximum number of test intervals is given in a more natural way.

The number of test intervals needed to describe the complete approximated request bound function is bounded, depending only on the number of tasks and the error. So, using the approximated functions instead of the exact function for analysis will automatically bound the number of test intervals. Using again the exceeding costs we can do the analysis with the algorithm 14. It is a small extension of the algorithm for the non-arbitrary case.

The version for event streams is given in algorithm 15.

5.4. Dynamic adaptive test

To improve the run-time of the exact test, an adaptive analysis is proposed. The idea behind this is to use the approximation for skipping as many test intervals as possible. We assume that there are only few parts of the functions having a short distance and therefore require to be analyzed exactly. We use the same ideas as for the efficient dynamic priority scheduling analysis.

Algorithm 14 Exceeding-Cost Approximation II

```
Algorithm Exceeding-Cost-Approximation (arbitrary case)
Given: task set Γ, τ, k
IF U_Γ = Σ_{∀τ∈Γ} c_τ^+ / p_τ > 1 ⇒ not schedulable
Δt_test := d_τ
WHILE (Δt_test ≤ k p_τ − j_τ)
  testlist := {}
  approxlist := {}
  U_r := 0
  Δt_old := Δt_test
  C_r = δ(Δt_old, τ, k) + ρ(Δt_old + j_τ, Γ_hp(τ), k) − χ(Δt_old)
  FOR ALL τ' ∈ Γ
      IF (p_τ' k − j_τ' ≤ Δt_old)
          ADD τ' TO approxlist
       ADD te = ((p_τ' k − j_τ'), τ) TO testlist
       U_r := U_r + c_τ'^+ / p_τ'
   ELSE
       ADD te = ( ( ⌊ (Δt_old + j_τ') / p_τ' ⌋ p_τ' − j_τ' ), τ' ) TO testlist
   END IF
  END FOR
  IF (Δt_test = d_τ)
      Δt_test := d_τ + p_τ − j_τ
  ELSE
      Δt_test := Δt_test + p_τ
  END IF
  WHILE (testlist ≠ {} AND C_r > 0)
   te = TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
   Δt_act = INTERVAL OF te
   τ' = TASK BELONGING TO te
   REMOVE te FROM testlist
   C_r := C_r − (χ(Δt_old) − χ(Δt_act)) − U_r(Δt_old − Δt_act)
   IF (τ' ∈ approxlist) C_r = {})
       REMOVE τ' FROM approxlist
       U_r := U_r − c_τ'^+ / p_τ'
   ELSE
       C_r := C_r − c_τ'^+
   END IF
   IF (Δt_act − p_τ' < 0 ∧ Δt_act > 0)
       ADD te := (0, τ') TO testlist
   ELSE IF (Δt_act > 0)
       ADD te := (Δt_act − p_τ', τ') TO testlist
   END IF
  Δt_old := Δt_act
  END WHILE
  IF (C_r > 0)
   ⇒ not schedulable
  END IF
END WHILE
⇒ schedulable
```

Algorithm 15 Exceeding-Cost Approximation II - Event Stream Version

```
Algorithm Exceeding-Cost-Approximation (arbitrary case)
                                        Event-Stream-Version
Given: task set Γ, θ, k
IF U_Γ = Σ_{∀τ∈Γ} Σ_{∀θ∈Θ_τ} c⁺_τ / p_θ > 1 ⇒ not schedulable
τ := τ_θ
Δt_test := d_τ
WHILE (Δt_test ≤ a_θ + k p_θ)
  testlist := {}
  approxlist := {}
  U_r := 0
  Δt_old := Δt_test
  C_r = δ(Δt_old, θ, k) + ρ(Δt_old, Γ_hp(τ), k) − χ(Δt_old)
  FOR ALL ∀τ' ∈ Γ_hp(τ) ∀θ' ∈ Θ_τ :
     IF (a_θ' + p_θ' k ≤ Δt_old)
        ADD θ' TO approxlist
      ADD te = ((p_θ' k + a_θ'), θ) TO testlist
      U_r := U_r + c⁺_τ' / p_θ'
   ELSE
      ADD te = ( ( ⌊ (Δt_old − a_θ') / p_θ' ⌋ p_θ' + a_θ' ), θ ) TO testlist
   END IF
  END FOR
  Δt_test := Δt_test + p_τ
  WHILE (testlist ≠ {} AND C_r > 0)
   te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
   Δt_act =INTERVAL OF te
   θ' := EVENT ELEMENT OF te
   τ' :=TASK BELONGING TO θ
   REMOVE te FROM testlist
   C_r := C_r − (χ(Δt_old) − χ(Δt_act)) − U_r(Δt_old − Δt_act)
   IF (θ' ∈ approxlist) C_r = {})
      REMOVE θ' FROM approxlist
      U_r := U_r − c⁺_τ' / p_θ'
   ELSE
      C_r := C_r − c⁺_τ'
   END IF
   IF (Δt_act > 0)
      ADD te := (Δt_act − p_θ, τ) TO testlist
   END IF
   Δt_old := Δt_act
  END WHILE
  IF (C_r > 0)
   ⇒ not schedulable
  END IF
END WHILE
⇒ schedulable
```

Algorithm 16 Adaptive test for static priorities with non-arbitrary deadlines

```
Algorithm static-adaptive
Given: task set Γ, τ
IF U_Γ = Σ_{∀τ∈Γ} c_τ^+ / p_τ > 1 ⇒ not schedulable
testlist := {}
approxlist := {}
Δt_old := d_τ
∀τ ∈ Γ: ADD te = ( ( max( 0, ⌊(Δt_old − a_τ)/p_τ⌋ p_τ + a_τ ) ), τ ) TO testlist
cost := ρ(Δt_old, Γ_hp(τ)) + c_τ^+
service := χ(Δt_old)
WHILE (testlist ≠ {})
    te=TEST LIST ELEMENT WITH SMALLEST Δt IN testlist
    Δt_act =INTERVAL OF te
    τ =TASK BELONGING TO te
    REMOVE te FROM testlist
    exCosts :=
exCosts + c_τ^+ − (χ(Δt_old) − χ(Δt_act)) + U_r(Δt_old − Δt_act)
    WHILE (exCosts > (costs − service))
        IF (approxlist = {})
            ⇒ scheduable
        τ' = FIRST ELEMENT OF approx-list
        exCosts := exCosts − ( ⌊(Δt_act − a_τ')/p_τ'⌋ + 1 − (Δt_act − a_τ')/p_τ' ) c_τ'^+
        ADD te := ( ( max( 0, ⌊(Δt_act − a_τ')/p_τ'⌋ p_τ' + a_τ' ) ), τ' ) TO testlist
        REMOVE τ' FROM approxlist
        U_r := U_r − c_τ'^+ / p_τ'
    END WHILE
    U_r := U_r + c_τ^+ / p_τ
    τ ADD TO approxlist
    Δt_old := Δt_act
END WHILE
⇒ not schedulable
```

Unfortunately, we cannot use the approximated request bound function for such an analysis. The reason is that only a few intervals $\Delta t'$ fulfill the condition of lemma 5.1.6, for the remaining intervals the calculated value exceeds the available capacity. As the approximated request bound function is always equal or larger than the exact request bound function, it could likely be that for some of these intervals the approximated request bound function remains above the capacity despite that the exact one steps below the capacity. These intervals would be missed by a dynamic analysis using the above approximated request bound function.

It is necessary to underestimate the exact value of the request bound function to guarantee that an existing interval $\Delta t'$ is found by the adaptive test. Using the exceeding cost approach we can achieve an adaptive test for static priorities. We will use an approximation of the exceeding cost bound function. With this function we still can overestimate the costs. The algorithm for this analysis for non-arbitrary deadlines is given in algorithm 16.

For the arbitrary deadline case we need to consider the algorithm for each job within the first busy period of each task.

5.5. Complexity

The complexity of the new static priority analysis is unknown. We can only give a few provisions. The complexity has to cover at least the calculation of the busy period and the maximum number of tasks within it. The busy period is calculated, as the worst-case response-time, with a fixed-point analysis. The busy period is the smallest value of Δt for which the following condition holds:

$$\min(\Delta t | \Delta t \geq \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau} \left\lceil \frac{\Delta t - a_\theta}{p_\theta} \right\rceil c_\tau^+)$$

The length of this busy period cannot be bounded in polynomial time, same as the calculation complexity. Therefore the arbitrary analysis has a pseudo-polynomial or exponential complexity at least out of these conditions.

For the non-arbitrary case it is only necessary to consider the first job for each task. The complexity for doing this can be bound by the maximum possible number of calculation points for the exceeding cost function. In the worst case, to proof a task τ, we have to consider to all jobs of all tasks (with equal and higher priority than τ) occurring within $d_{\tau,1}$. As each of these jobs can be the last job required to fulfill the schedulability condition of theorem 5.2.4 we need to calculate the exceeding cost function step-wise. The contribution of each job is considered separately. A specific job can either contribute to the exceeding costs or reduce the exceeding cost function. Again, the number of jobs from tasks with small periods and deadlines within the deadline of a task with a large period and deadline depends on the ratio between the larger and the smaller periods and deadlines. Therefore the number of occurring jobs of higher priority tasks cannot be bound in general and can become quite large. Therefore the upper bound for the complexity is pseudo-polynomial.

For the approximative analysis of course a smaller bound on the complexity can be given. The approximation error bounds the number of test intervals that have to be considered exactly for each task and each of the involved functions separately. During the analysis no more test intervals have to be considered, even in the arbitrary case. As we do the analysis for each task separately we can bound the overall complexity for the approximative analysis by $O(n^2 \ln n \frac{1}{\varepsilon})$.

To estimate the real effort required for the analysis we use again tests with randomly generated task sets. We compare the effort for the analysis with the response time analysis in the efficient implementation by Sjödin and Hansson [122]. The real effort needed for the new analysis is quite lower than the theoretical bound. The experimental results are given in chapter 6.

CHAPTER 6

Evaluations

To evaluate the proposed tests and algorithms and to learn about their strengths and weaknesses we have done a set of experiments with randomly generated task sets. A task set has several characteristic parameters like the number of tasks, utilization, the ratio between the largest and smallest period in the task set and so on. In our experiments we investigate task sets with different parameters and we show how the variation of just one parameter affects the run-time and the recognition rate of the various analysis algorithms.

We have not only implemented our new algorithms but also many of the existing algorithms to compare them with our results under the same experimental conditions.

6.1. General setup of the experiments

In the following we will describe the setup of our experiments.

6.1.1. Technical setup. All experiments where performed on a computer with two Intel 3 Ghz Quad-Core processors but using only one core for each algorithm. The experimental framework and the algorithms where implemented in Java.

6.1.2. Generation of random task sets. In the following we will explain in detail the generation of random task sets behind all our experiments. First we have to choose for each task set the number of tasks, either randomly or by selection.

For each task τ we have to choose randomly:

- The period p_τ
- the worst-case execution time c_τ^+
- the deadline d_τ

There exist dependencies between the values of a task and between the values of the different tasks in a task set. These dependencies have to be taken into account to achieve realistic task sets and task sets close on the boarder between schedulability and non-schedulability. Therefore our random task-set generation has the following steps:

(1) Choose (or draw) a number of tasks
(2) Choose (or draw) the total utilization of the task set U_Γ. For uni-processor systems the utilization has to be smaller or equal to 100%.
(3) Distribute the utilization on the tasks of the task set U_τ. We present the available algorithm for a realistic distribution in section 6.1.3.
(4) Draw for each task a period p_τ.
(5) Calculate the worst case execution time c_τ^+ for each task τ from the chosen period p_τ and specific utilization U_τ. $(c_\tau^+ = p_\tau U_\tau)$

Algorithm 17 UUniFast Algorithm [24]

```
Algorithm UUniFast
sumU := U_Γ
nextSum := 0
for i from 1 to |U_Γ|
    nextSum := sumU · rand(1/(n−i))
    U[i] := sumU − nextSum
    sumU := nextSum
end for
return U[];
```

(6) Draw a deadline for each task. The deadline of a task has to be larger than its worst-case execution time. For uni-processor systems with EDF or fixed priority scheduling without jitter only those deadlines are relevant which are smaller than the corresponding period. Therefore we distribute the deadline somewhere between the worst-case execution time and the period of a task. We call the ratio of the deadline and the period of a task the gap of the task. For the experiments an average value for this gap can be set. The gap is measured in percent of the period of the task. A task with for example a period of 100 ms and a deadline of 30 ms has a gap of 70%, a task with a period of 50 ms and a deadline of 40ms has a gap of 20% and a task with a period of 10 s and a deadline of 5 s has a gap of 50%. Tasks with a small gap are more likely to be schedulable than tasks with a large gap.

6.1.3. Distribution of utilization. There are several ways to distribute the utilization of a task set on its tasks, see [24], [26] for a complete discussion of the problem. In the following we will shortly summarize the different possible approaches introduced there. The first approach (UScaling) is to simply draw for each task $\tau \in \Gamma$, one after another, a part of the remaining utilization. In this approach the first task τ_1 gets a utilization between zero and U_Γ, the second task between zero and $U_\Gamma - U_{\tau_1}$ and so on. The last task gets the remaining utilization. Also the resulting utilization is the requested one, the probability is quite high that the first tasks gets nearly the whole remaining utilization which leaves nearly nothing for the remaining tasks. The probability to achieve a task set with a few large tasks (the first ones) and many tasks sharing only a small fraction of the utilization is quite high.

The second approach is to draw for each task independently a value (between 0 and 1 for example), add these values and scale the result with the total utilization of the task set. In this algorithm each task has an equal chance for the same fraction of utilization but with a high probability no tasks with a large utilization will occur. For example in a task set with 100 tasks the probability for individual tasks with a specific utilization of much more than 2% is quite low.

The problem is to generate realistic and non-trivial distributions. In [24] an efficient algorithm was proposed to generate a realistic distribution, the UUniFast algorithm. It is given in algorithm 17. The algorithm is an efficient implementation of the second approach

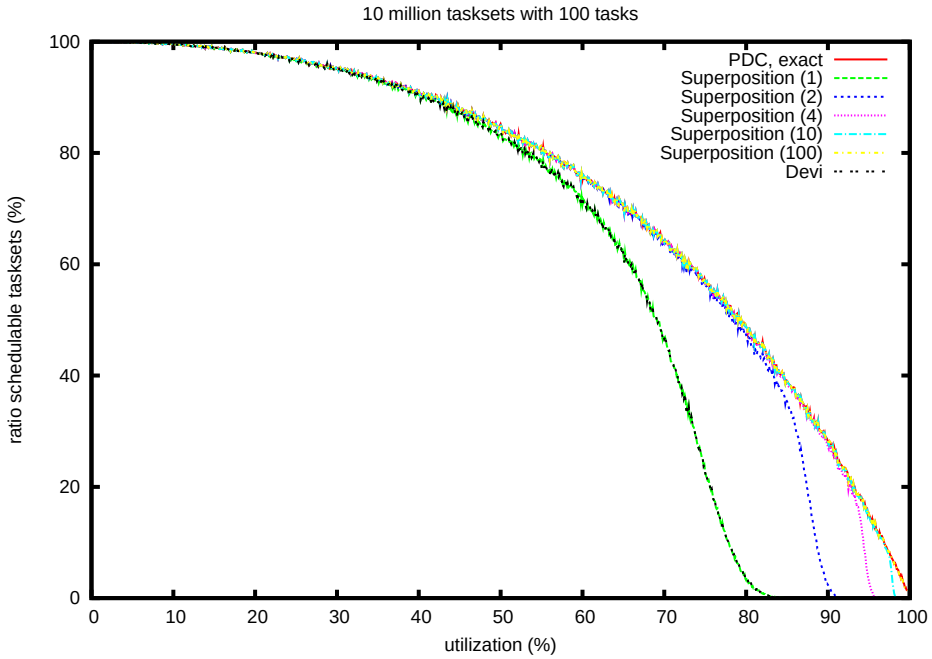

FIGURE 6.2.1. Superposition: ratio of schedulable task sets for different utilizations

but using an exponential distribution which allows large specific utilizations even for task sets with many tasks. This algorithm is used in the following experiments.

6.2. Superposition approximation

First of all we will focus on the superposition approximation for one-processor systems with EDF scheduling as introduced in chapter 3.

6.2.1. Setup. In the first experiment we compare the ratio of schedulable task sets versus all generated task sets for a specific utilization. On the x-axis we have the utilization of the tasks set running from 0% to 100%, on the y-axis we have the proportion of task sets recognized as schedulable on all task sets generated with this utilization. The gap between period and deadline has an average value between 5% and 95% of the period and both the period and the gap is chosen using a normal distribution. The periods have a value between 10 ns and 10 s. For the experiment we generated 10 million task sets with 100 tasks each and a utilization between 1% and 99%.

We analysed these task sets using the superposition approximation with one, two, four, ten and 100 exactly considered test intervals for each task, using the exact processor demand criterion (PDC) and using the previous best sufficient analysis, the test of Devi [46]. Note, that the drawn task sets do not contain tasks having a deadline smaller than their worst-case execution time.

6.2.2. Results. Figure 6.2.1 shows the ratio of schedulable task sets detected by each of the analysis algorithms for the first experiment. Note that all analysis algorithms run on

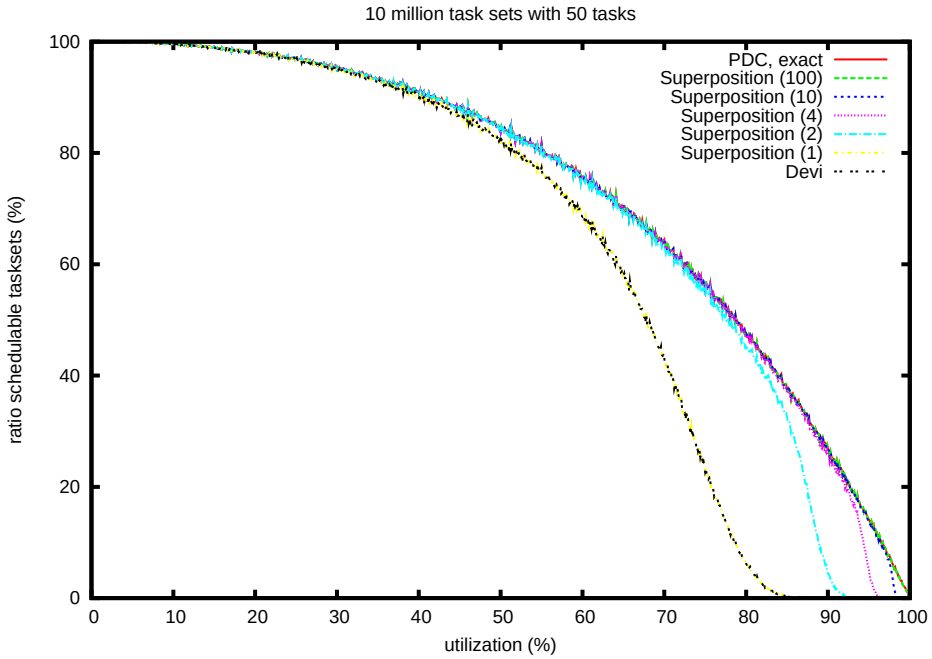

FIGURE 6.2.2. Superposition: ratio of schedulable task sets (50 tasks)

the same task sets. The difference between the ratio of schedulable task sets for a specific approximation degree and the ratio for the exact processor demand criterion (PDC) is a good indication for the quality of the approximation and helps for a reasonable choice of the approximation degree.

In figure 6.2.2 the results for an experiment with the same setup as above but using task sets with 50 tasks is depicted. For figure 6.2.3 the same experiment with one million task sets with 500 tasks each is used. The setup for figure 6.2.4 is again the 10 million task sets with 100 tasks but the figure shows the ratio in dependency of the average gap instead of the utilizations.

A higher number of exactly considered test intervals (k) is leading to a higher ratio of schedulable task sets. As we can see the ratio for each superposition approximation is in all figures between the sufficient test of Devi [46] and the exact processor demand test. The approximation with $k = 1$ is equivalent to the test of Devi and therefore leads to the same ratio of schedulable task sets (see section 3.7). The approximations with higher values for the number of exactly considered test intervals close the gap between the test of Devi and the exact analysis quite fast. In these experiments more than 50% of those task sets being schedulable and which have not been recognized as schedulable by the test of Devi are recognized as schedulable by the next better approximation with $k = 2$. Values for k larger than 10 are sufficient to classify nearly all schedulable task sets correctly. The approximation with $k = 100$ classifies all task sets correctly in this experiment. Task sets not classified correctly as schedulable are mainly those having a high utilization. Despite that the test of Devi fails to classify nearly the entire schedulable task sets with a utilization of 80%, the approximation with $k = 2$ classify nearly all of them as schedulable. Up to a

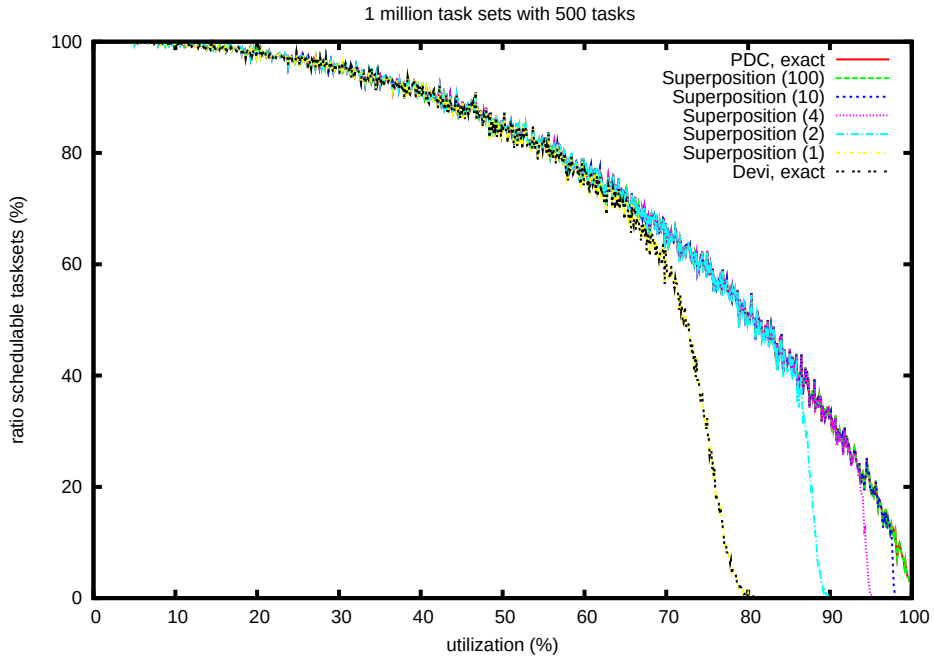

FIGURE 6.2.3. Superposition: ratio of schedulable task sets (500 tasks)

utilization of 85% nearly all classifications of the approximation with $k = 2$ are correct, for higher utilization the correct-classification rate drops fast and for the schedulable task sets with a utilization of more than 90% nearly non are classified as schedulable by this approximation. The approximation with $k = 4$ is good enough for task sets with a utilization up to 90% but fails for utilizations of more than 95%, the approximation with $k = 10$ is working satisfactorily for utilizations up to 98% and for an approximation with $k = 100$ there are nearly no wrongly classified task sets in this experiment. Comparing the figures it can be seen that the correct-classification rate is a bit higher for task sets with few tasks than for task sets with many tasks.

The figure 6.2.4 depicts for the experiment of figure 6.2.1 (10 million task sets with 100 tasks in each task set) the dependency of the acceptance ratio of the different algorithms and the average gap between deadline and period of a task. It is measured in percent of the period of the task. A task with a period of 100 ms and a deadline of 30 ms has a gap of 70%. Of cause, tasks having a small average gap between deadline and period and therefore a large average gap between worst-case execution time and deadline are more likely to be schedulable than other tasks. Therefore of the tasks with an average gap of 5% nearly 80% are schedulable whereas of the tasks with an average gap of 95% less than 65% are schedulable. The utilizations of the task sets are between 5% and 99%. The probability for a wrong classification is nearly equal for all values of the average gap for each of the algorithms and approximation errors. It is a bit lower for task sets with a small gap. The reason for this is that most of schedulable task sets with very high utilizations have a small value for the average gap and, as we have seen, those task sets are hard to distinguish for

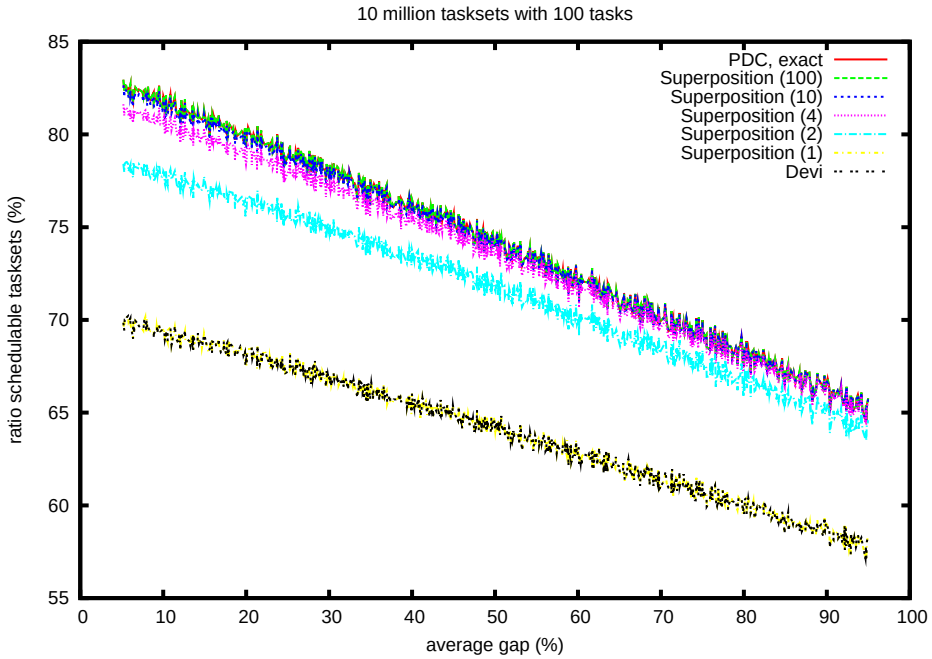

FIGURE 6.2.4. Superposition: ration of schedulable task sets for different average gaps

the approximation algorithms. The effect of approximation seems to be independent of the average gap alone.

In figure 6.2.5 the result of an experiment measuring the dependency of the acceptance rate on the ratio between the shortest and largest period of any task within one task set is presented. A task set having as smallest period of tasks the period 100 ms and as largest period of tasks the period 10.000 ms has a ratio of 100. The acceptance rate seems to be for all algorithms independent of the ratio between the shortest and largest period.

6.2.3. Results for analyses runtime. Let us now consider the run-times of the analyses algorithms. This experiment shows the quality of the approximation and therefore whether the approximation is suitable for certain applications. We have measured the worst-case and the average runtime in dependency of certain criteria of the task set, like the utilization, the ratio between the smallest and largest task, the gap and of course the number of tasks in a task set. We have calculated for each analysis the average and the maximum runtime required by the analysis algorithm in dependency of the chosen criteria for the previous experiments.

In figure 6.2.6 the average computation time for each of the approximation algorithm in dependency of the utilization is depicted and compared with the average effort for the processor demand criterion and the test of Devi [46]. The task set size was 100 tasks and the gap again between 5% and 95% with a normal distribution. The approximation with $k = 2$ requires only about 15% more effort than the test of Devi, despite that it has a much better recognition rate. The approximation requires in the average less than 30% of the

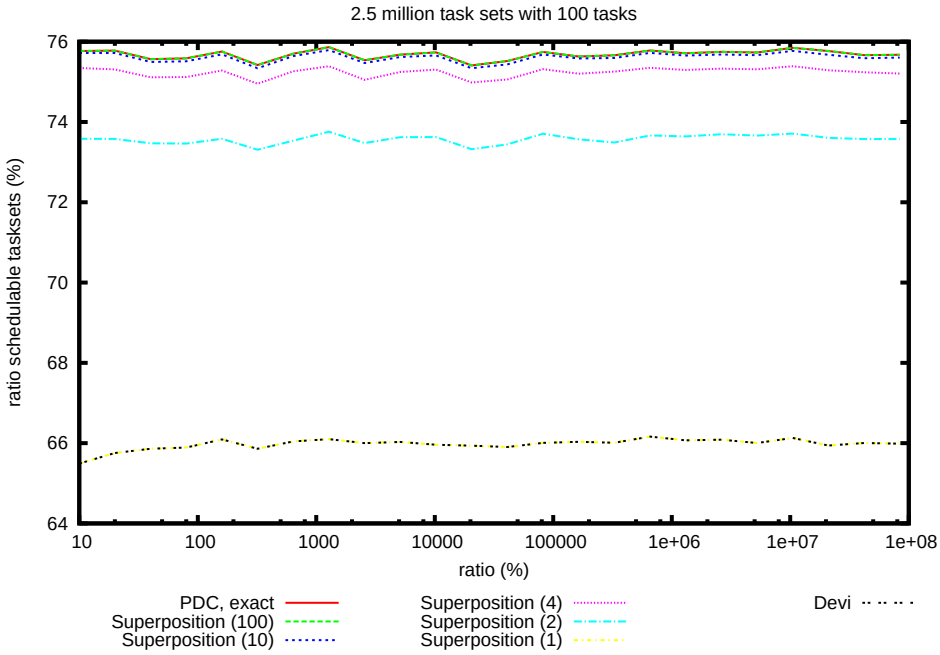

FIGURE 6.2.5. Superposition: ratio of schedulable task sets for different ratios between the largest and smallest task in the task set (100 tasks per task set)

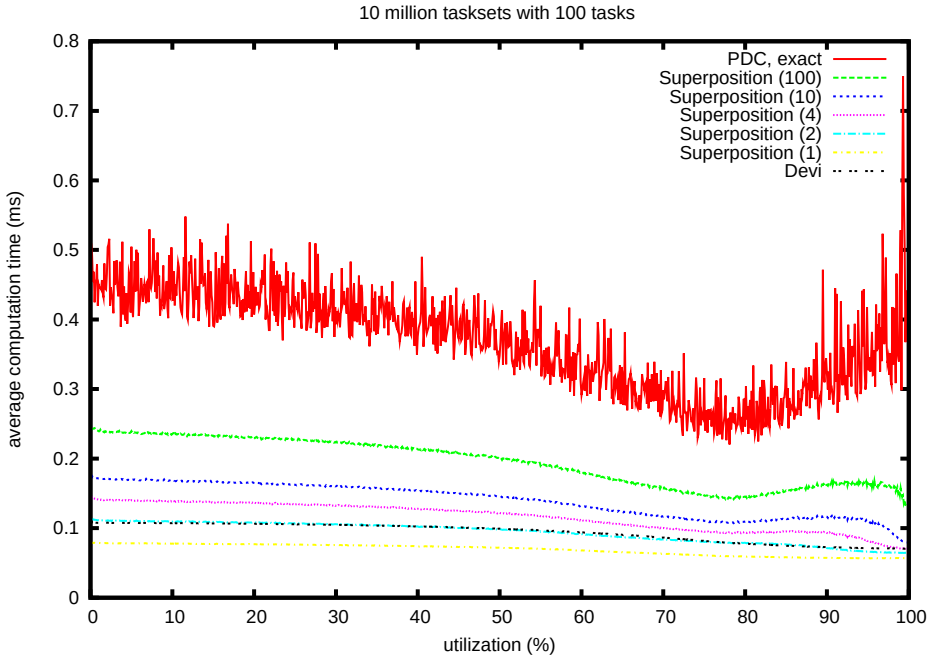

FIGURE 6.2.6. Superposition: average run-time for different utilizations

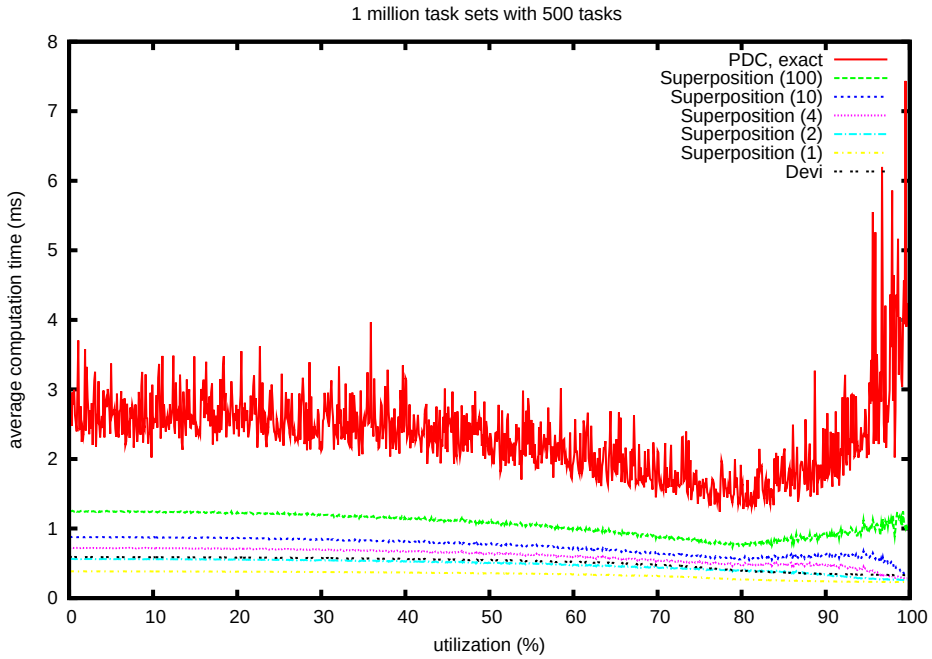

FIGURE 6.2.7. Superposition: average run-time for different utilizations (500 tasks)

effort required for the processor demand criterion. Even the approximation with $k = 100$ requires only about half of the effort of the processor demand criterion (but about twice as much as the test of Devi).

Another interesting point is that the variation of the effort is much higher for the processor demand criterion than for the approximations. For the algorithms the average computation time starts on a highest level at 5% and decreases slowly until about a utilization of 80%. For higher utilizations the effort rises again but declines for utilizations of nearly 100% for all approximations. The reason for these results is mainly the falling degree of schedulability with increasing utilization. On the one side in the average a non-schedulable tasks set requires not much effort to be classified as non-schedulable by the tests. Therefore the average effort declines with rising utilization. But on the other side the effort depends on the utilization as the maximum test interval depends on $\frac{U}{1-U}$. Therefore the effort for schedulability analysis increases for many task sets. As $\lim_{U \to 100\%} \frac{U}{1-U} = \infty$ the effort will increase faster when the utilization gets closer to 100%. By about 80% this increasing of the effort becomes larger than the decreasing of the effort as of the lower schedulability rate, therefore the overall effort starts rising again. For all approximations the recognition rate declines fast after reaching a certain level of utilization and therefore the effort declines too. Note, that, in contrary to the processor demand test, for the approximations the maximum effort is limited by the value k. In figure 6.2.7 the average effort for 1 million task sets with 500 tasks each is depicted.

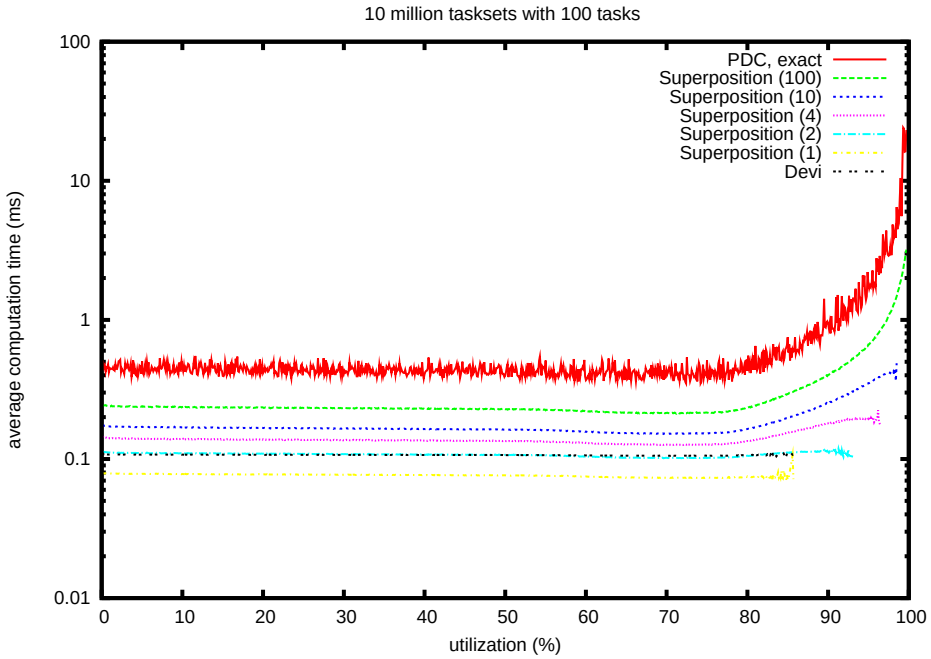

FIGURE 6.2.8. Superposition: average run-time for different utilizations for only the schedulable task sets

Figure 6.2.8 shows the average effort for only the schedulable task sets. The effort is comparable but does not decline before 80% utilization as it is the case for the average effort for all task sets.

More interesting than the average computation time is the worst-case for which we have measured the maximum computation time. Figure 6.2.9 shows the maximum effort for the superposition approximation with a k of 10 and 100 and the processor demand criterion compared again with the test of Devi. Note that this figure uses a logarithmic scale.

The measured approximations with $k = 2$, $k = 4$ and $k = 10$ have a maximum effort between 0.250 ms and 1 ms for most of the utilizations, only the approximation $k = 10$ requires once 1.75 ms. The effort for the approximation with $k = 100$ is a bit higher for most utilizations and rises significantly to up to 8 ms for very high utilizations. Only for these utilizations do the approximations require their available budget. In the other cases the number of really required test intervals is much lower than the number of allowed test intervals by k.

In figure 6.2.10 the maximum effort for 50 tasks and in figure 6.2.11 the maximum effort for 1 million task sets with 500 tasks each is depicted.

Compared to the maximum effort required for the processor demand test the effort required for the approximations is very low. The effort for the processor demand test is between $50\,ms$ and $500\,ms$ for most of the utilizations rising to up to $1.5\,s$ at the maximum. Note, that we have allowed periods between 10 ms and 10 million ms here. Increasing or

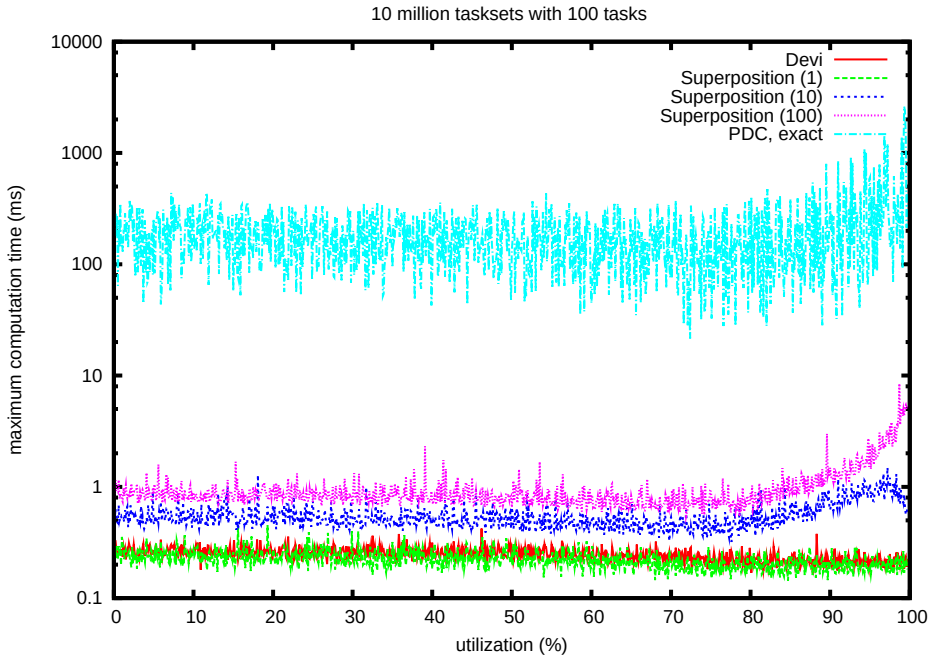

FIGURE 6.2.9. Superposition: maximum run-time for different utilizations (with PDC)

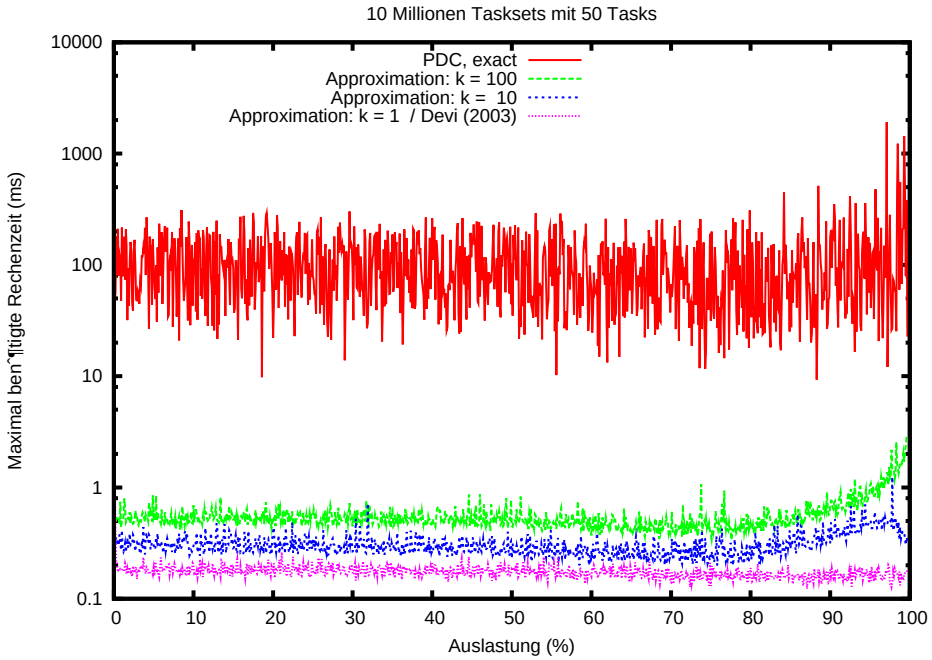

FIGURE 6.2.10. Superposition: maximum run-time for different utilizations (50 tasks)

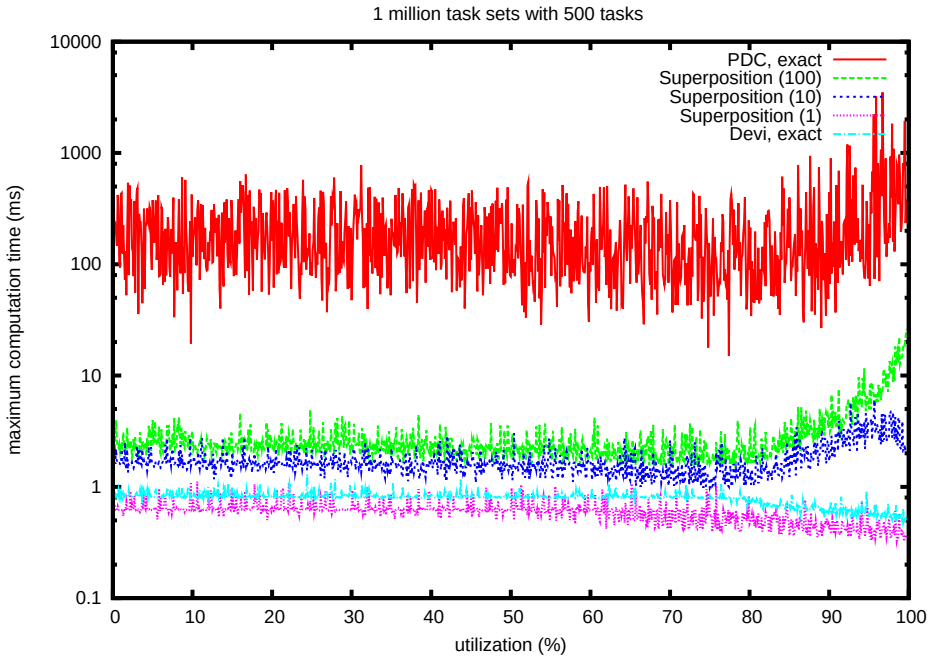

FIGURE 6.2.11. Superposition: maximum run-time for different utilizations with PDC (500 tasks)

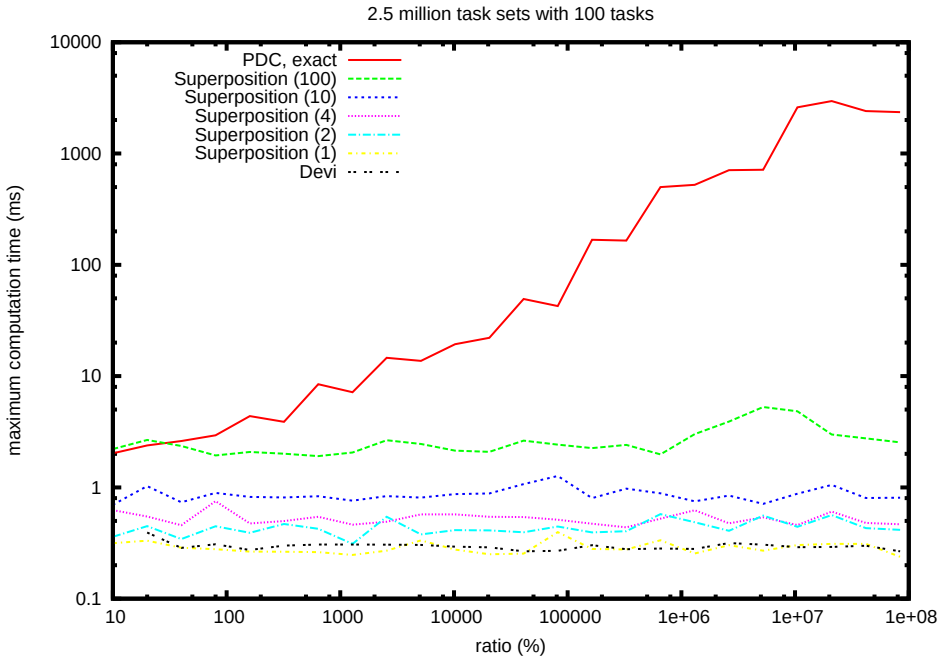

FIGURE 6.2.12. Superposition: maximum run-time for different ratios between smallest and largest task in the task set

reducing this value has a significant impact on the required computational effort for the processor demand criterion, as we will show in the next experiment.

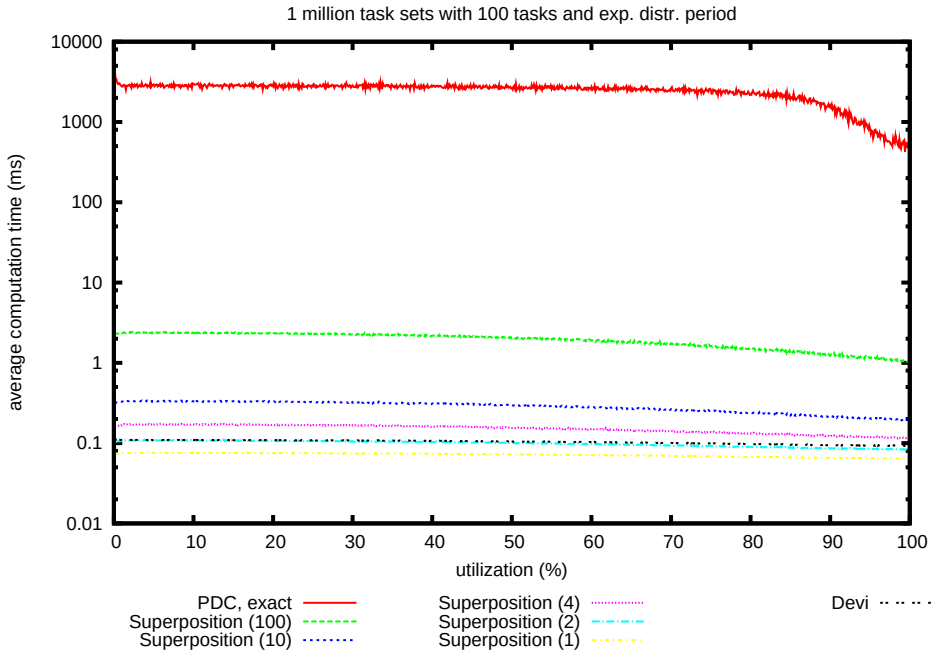

FIGURE 6.2.13. Superposition: Average run-time for different utilizations with exponential distribution of periods

In figure 6.2.12 the dependency of the computation effort on the ratio between the smallest and largest task in the task set is shown. For this experiment we have used a normal distribution for the periods. The approximation algorithm and the sufficient test are independent of the ratio between smallest and largest task but the processor demand criterion depends clearly on the ratio. In figure 6.2.13 and figure 6.2.14 we have repeated the experiment with an exponential distribution of the periods. The advantage of the approximations is even higher in this case as the processor demand test requires more effort.

The last question is the dependency of the computational effort on the number of tasks. It is shown in figure 6.2.15. Of cause, all algorithms require a larger computation time with an increasing number of tasks. It seems that this increase is linear for all algorithms. For the superposition approximation this confirms the complexity of $O(n\log(n)k)$ which depends nearly linear on the number of tasks.

We have seen that the proposed approximations deliver good results even for small values of k and are much faster than the previous exact processor demand test. Compared to the sufficient test of Devi the run-time is larger but still acceptable, but the recognition rates are much higher especially for task sets with utilization between 80% and 95%.

6.3. Dynamic Approximation Approaches

In the following we will consider the run-time of the new adaptive tests introduced in chapter 4. We have proposed the dynamic-error test, which starts with a low degree of exactness and only adapt this degree as far as necessary to distinguish between schedulable and non-schedulable task sets and the all-approximation algorithm in which approximation

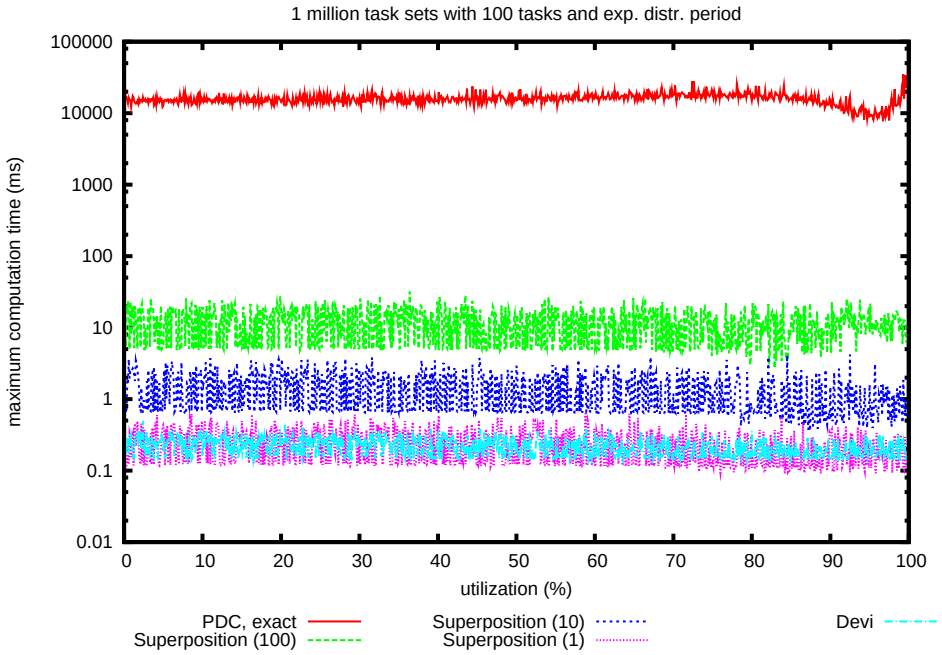

FIGURE 6.2.14. Superposition: maximum run-time for different utilizations with exponential distribution of periods

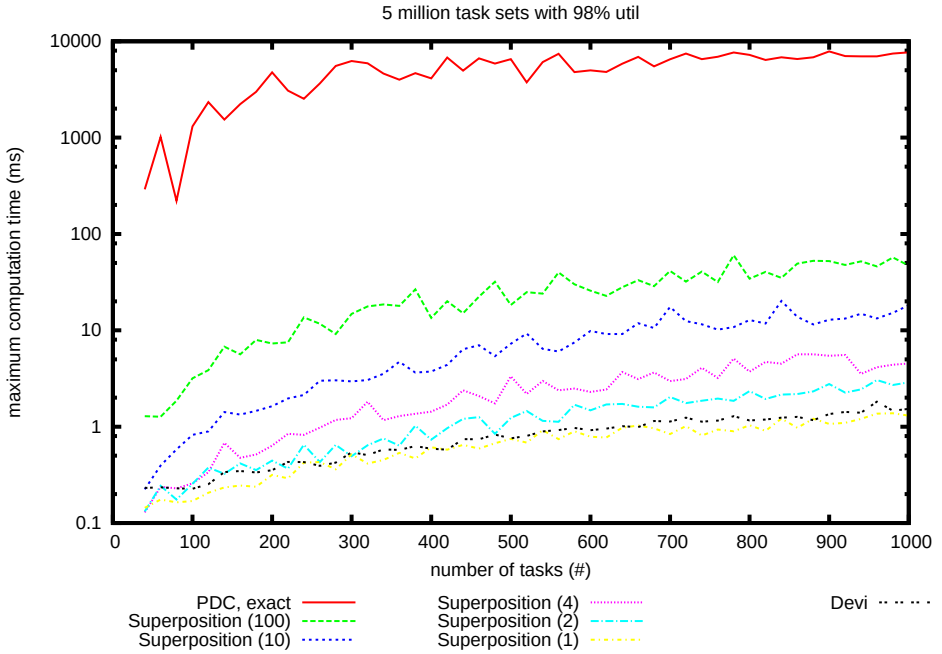

FIGURE 6.2.15. Superposition: run-time for different number of tasks in the task sets

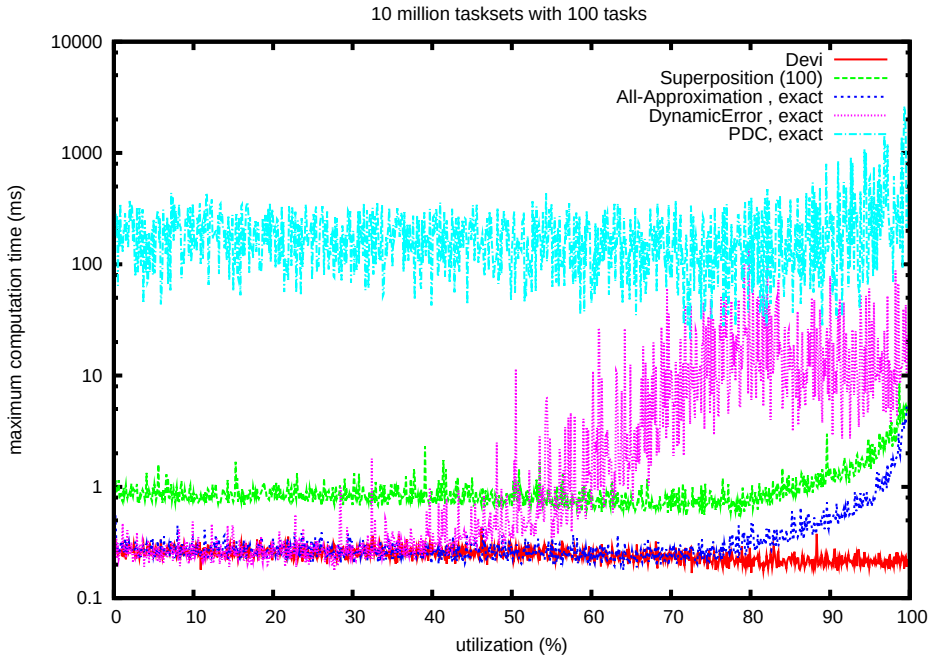

FIGURE 6.3.1. Adaptive analysis: maximum run-time

is used as much as possible. In the all-approximation algorithm all tasks are considered approximated and only for test intervals for which the test fails with this assumption the approximation of the tasks is removed task-by-task until the test does not fail for these test intervals any more.

6.3.1. Setup for utilization based generation of random task-sets. For the experimental setup 5 million task sets with 100 tasks each are generated to compare the run-time of the different algorithms. In figure 6.3.1 and figure 6.3.3 we have measured the maximum run-time required to analysis task sets with certain utilizations for all algorithms using again a logarithmic scale. The upper-most curve shows the required effort for the processor demand analysis (PDC), the second curve with the large fluctuation the effort for the dynamic error approximation. The third curve shows the superposition approximation with $k = 100$. The lowest curve is the test of Devi and the all-approximation algorithm is the curve that is equal to the test of Devi at the beginning and leads to a higher effort starting by about 75%.

6.3.2. Results for utilization-based generation of random task-sets. The success ratio is, of course, equal for all the tests. In the maximum the processor-demand test requires about 50-100 times more run-time than all adaptive analyses. This difference in run-time depends, as we will see later, on the allowed range of periods, which was 10 ns - 10 million ns here. In the run-time of the processor demand analysis and also the dynamic-error analysis we have a large fluctuation even for small changes of the utilization. The run-time of the all-approximation test seems to be relatively constant for small changes of the utilization. The dynamic-error analysis requires significantly more run-time. In this

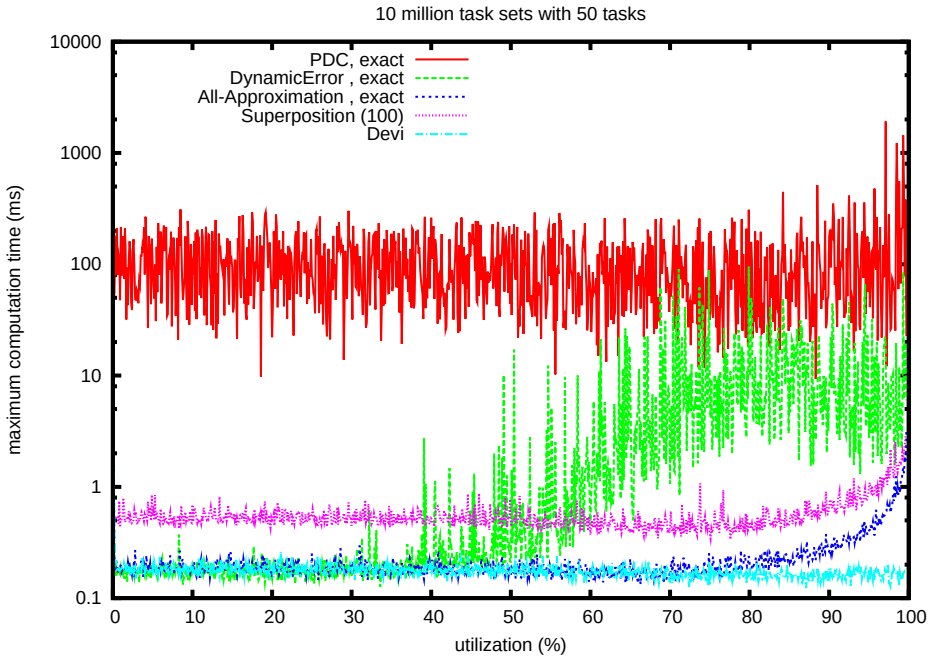

FIGURE 6.3.2. Maximum computation time of adaptive analysis (50 tasks)

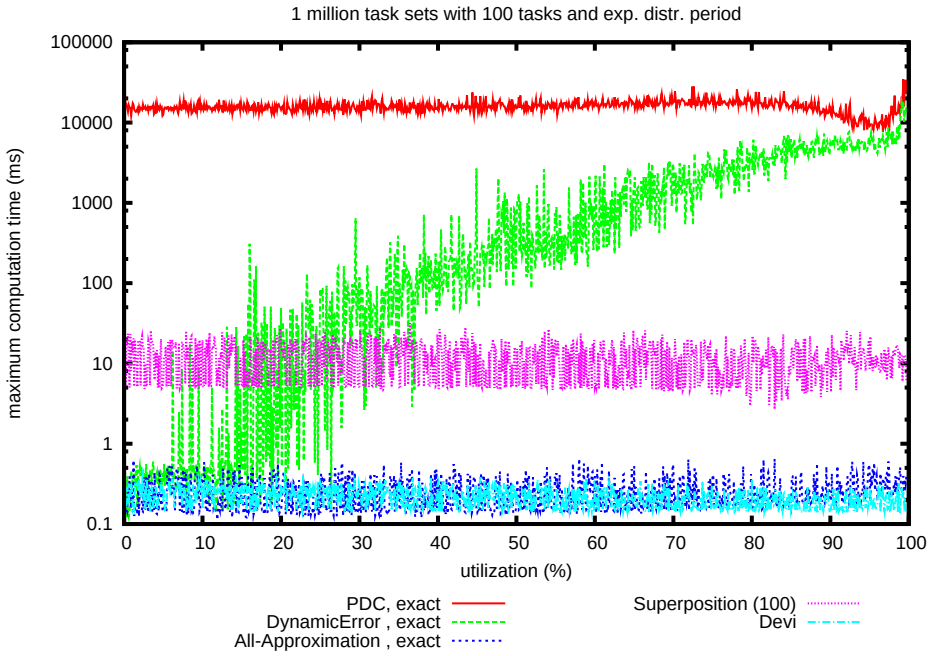

FIGURE 6.3.3. Adaptive analysis: maximum run-time - exponential distribution of periods

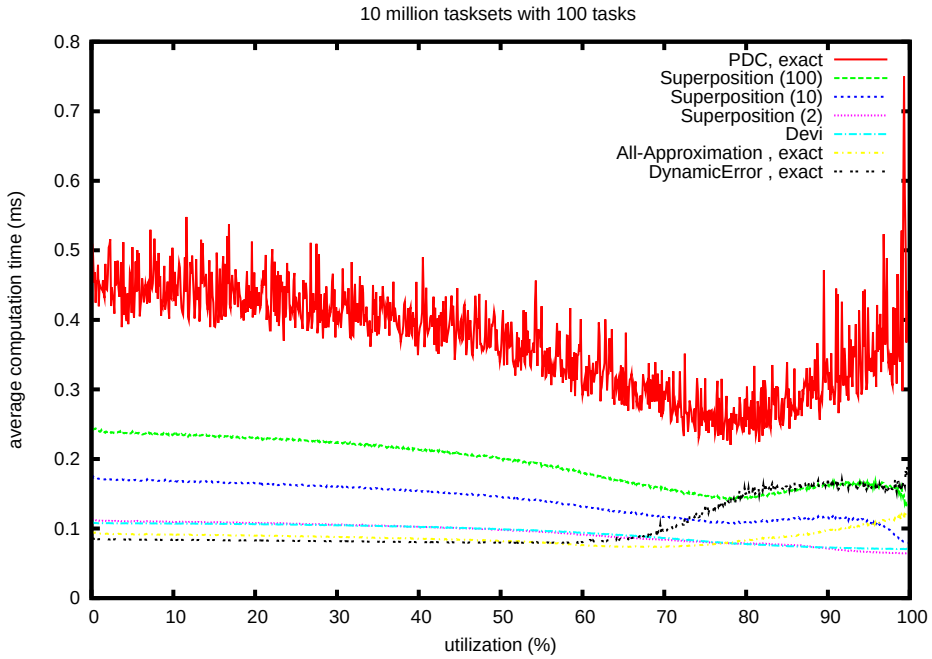

FIGURE 6.3.4. Adaptive analysis: average run-time

experiment the requirement for significant run-time starts at about 60% utilization and has a peek between 80% and 95% utilization. It seems that the concrete values for the tasks have a larger influence on the run-time than the utilization. In the area up to 95% utilization we have a significant level on schedulable task sets (as shown in figure 6.2.1) and therefore a large chance to have a task set in each drawing requiring a large run-time. The peeks of the run-time are somewhere between 20 ms and 140 ms compared with less than 1 ms for the all-approximation algorithm at the same utilizations levels. The run-time for the all-approximation test goes up to 27 ms in this experiment but is always much lower than the effort for the dynamic error test. For all-approximation analysis the run-time is in the same region as the run-time of the test of Devi for the utilization up to 75%. After this utilization the tests require significantly more run-time. With rising utilization the increase of the maximum required run-time gets larger and the run-time seems to explode for task sets with utilizations very close to 100%. But note that for such high utilizations the test of Devi cannot classify any schedulable task set correctly.

In figure 6.3.4 we have measured the average effort for the same experiment. The processor demand criterion starts between 0.4 ms and 0.5 ms for small utilizations and requires, after a short declining, run-times between 0.3 ms and 0.75 ms in the average for very large utilizations. The all-approximation algorithm instead requires less than 0.1 ms for small utilizations and rises to 0.16 ms in the average for large run-times.

In the figures 6.3.5 and 6.3.6 we have depicted the maximum respectively average run-time for 1 million task sets and with 500 tasks each. The curves looks similar but the absolute values for the curves and the distance between the curves become larger.

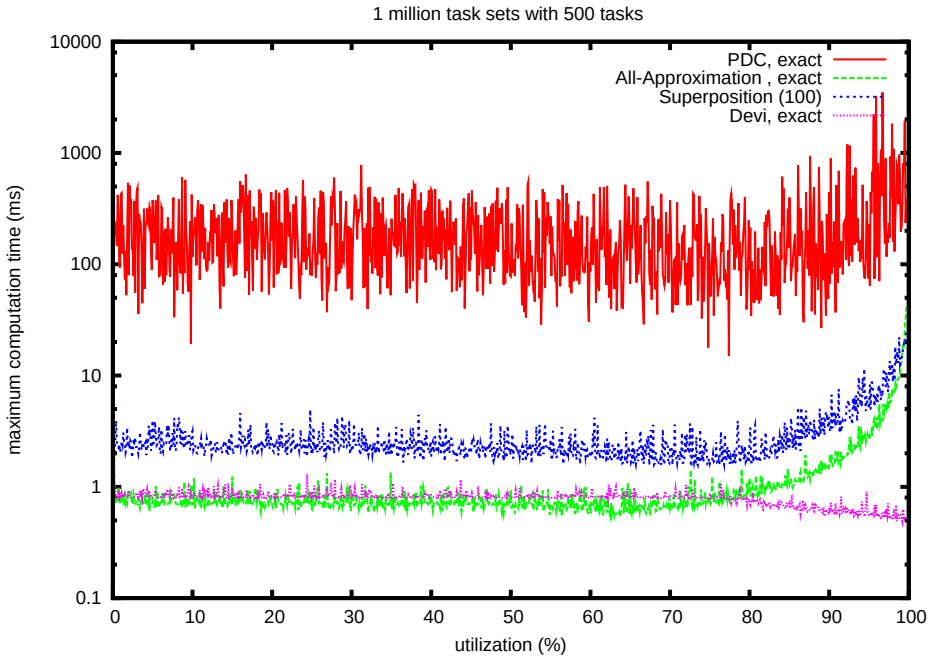

FIGURE 6.3.5. Adaptive analysis: maximum run-time (500 tasks)

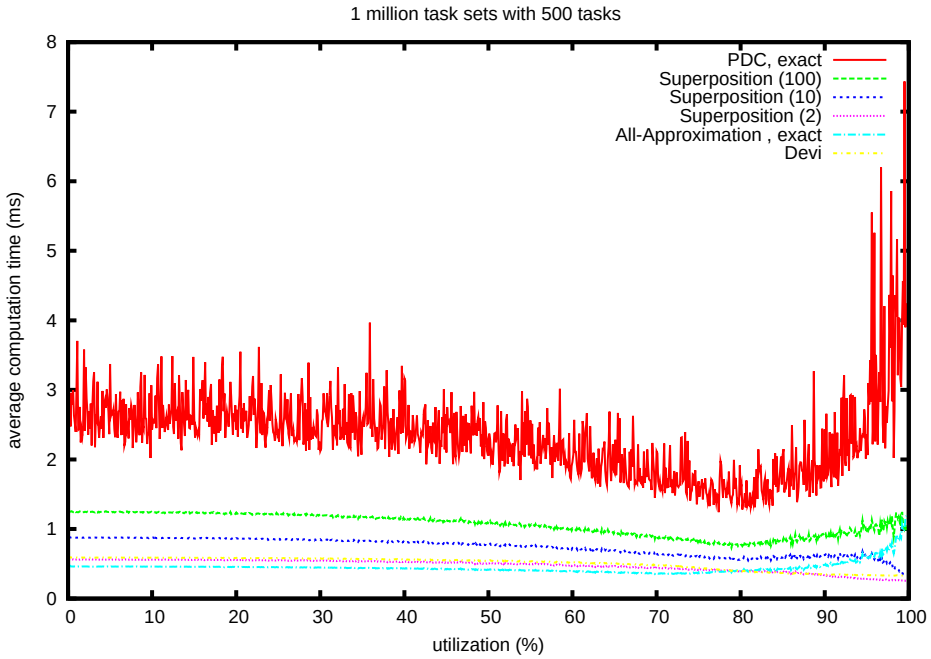

FIGURE 6.3.6. Adaptive analysis: average run-time (500 tasks)

6.3.3. Setup for ratio-based generation of task sets. In the next experiment (figure 6.3.7) we have evaluated the dependency of the run-time on the ratio between tasks with small and large parameters within the task sets. We have considered ratios between the smallest period of any task and the largest period of any task in the task set from 10 to

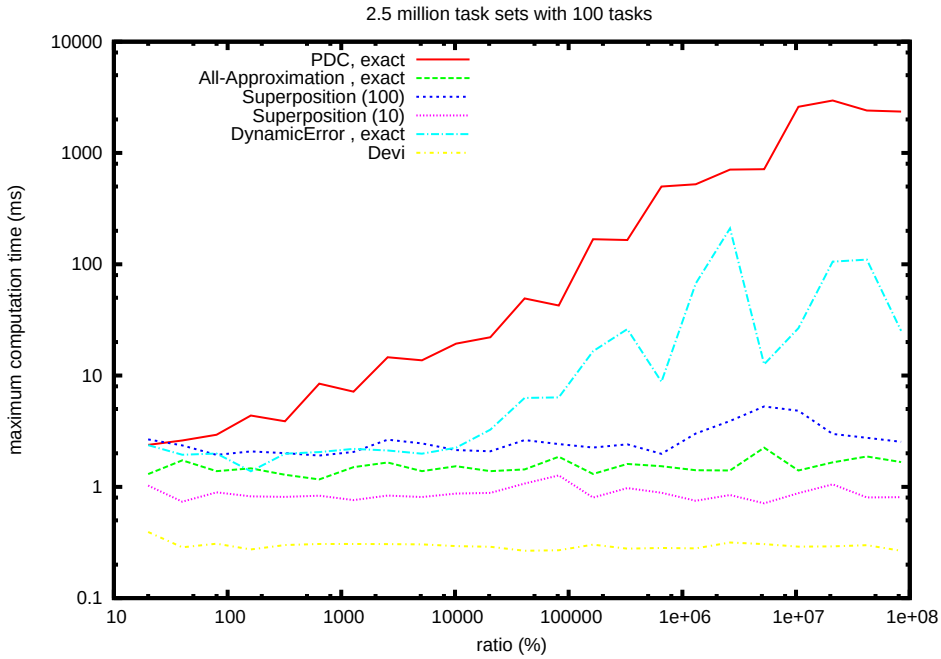

FIGURE 6.3.7. Adaptive analysis: maximum run-time for different ratios between largest and smallest period for 98% utilization

$\frac{max(p)}{min(p)}$	PDC	All Approx.	All-approx. period	all-approx. inverse
100	9.941 ms	0.620 ms	1.269 ms	6.912 ms
1,000	104.4 ms	0.768 ms	0.796 ms	17.268 ms
10,000	665.1 ms	0.650 ms	0.684 ms	8.860 ms
100,000	4.287 s	0.525 ms	0.637 ms	6.950 ms
1,000,000	22.03 s	0.521 ms	0.500 ms	6.198 ms
10,000,000	267.2 s	0.530 ms	0.773 ms	7.452 ms
100,000,000	1375.7 s	0.605 ms	0.554 ms	4.295 ms

TABLE 1. Adaptive analysis: Maximum execution time (ms) for 100 task and 98% utilization

100,000,000. We have done two experiments, one with a normal distribution of the period, but at least one task small enough to guarantee the ratio, the other with an exponential distribution. We have used a utilization of 98%.

6.3.4. Results for ratio-based generation of task-sets. The run-time of the processor demand test depends directly on the ratio and doubles when the ratio is doubled. The all-approximation test is independent of the ratio between the largest and smallest period of any task of the task set.

Table 1 and table 2 show the results of an experiment where the period of the task is exponentially distributed. Each task set has 100 tasks and a total utilization of 98% and we have generated 20,000 task sets for each ratio. The table 1 shows the measured maximum execution times for each ratio/ algorithm combination. The largest period in

$\frac{max(p)}{min(p)}$	PDC	All Approx.	All-approx. period	all-approx. inverse
10	0.121 ms	0.0986 ms	0.0956 ms	0.145 ms
100	0.261 ms	0.0902 ms	0.0881 ms	0.234 ms
1,000	1.050 ms	0.0907 ms	0.0884 ms	0.288 ms
10,000	5.544 ms	0.0915 ms	0.0887 ms	0.272 ms
100,000	34.53 ms	0.0925 ms	0.0894 ms	0.237 ms
1,000,000	237.2 ms	0.0934 ms	0.0902 ms	0.197 ms
10,000,000	1.985 s	0.0961 ms	0.0926 ms	0.173 ms
100,000,000	30.980 s	0.0999 ms	0.0957 ms	0.159 ms

TABLE 2. Adaptive analysis: average run-time (ms) for 100 task and 98% utilization

a task set is 10, 100, 1000, ..., 100 million times larger than the smallest period within the same task set. The execution times for the all-approximation algorithm and the all-approximation algorithm with an approximation queue sorted by the period of the tasks are mostly constant in the rage of 0.5 to 0.8 ms. Even using the inverse order for the approximation queue requires only 17 ms run-time at most. The processor demand test instead requires only for small ratios a comparable low execution time, for ratios in the area of 1 million it requires up to 22 seconds execution time and for ratios of 10 million or 100 million it requires 4 minutes respectively 22 minutes. The run-time grows a bit less than the ratio but still becomes unacceptably large very soon.

The table 2 shows the average execution times for all generated 20,000 task sets. Of course these values are all much smaller than the maximum execution times as the 20,000 task sets surly contain many “easy” task sets. For the all-approximation algorithms they are more constant than their maximum execution times with values between 0.09 ms and 0.10 ms for the standard respectively 0.15 ms to 0.3 ms for the inverse case. The effort for the processor demand criterion grows from 0,1 ms for a ratio of 10 to 31 seconds for a ratio of 100 million.

Therefore the run-time of the all-approximation algorithm can be bounded in the worst-case only with the utilization and the number of tasks, independently of the other parameters of the task set.

6.3.5. Dependency on the number of task in the task set. Figure 6.3.8 shows the dependency of the run-time on the number of tasks in the task set. Of course the run-time depends on the number of tasks. As we have to consider each task at least once to prove schedulability we have at least a linear increase of the required run-time. The measured increase is a bit higher than $O(n)$. The dependency can be bounded by $O(n \cdot \log(n)^{1.5})$, see the following section for details.

6.3.6. Variations of the all-approximation algorithm. The order of the tasks in which their approximation is removed when the test fails at a certain test interval can be important for the run-time of the test. For simplicity of the implementation we have considered only fixed orders. The first and most promising order is by the difference between deadline and period of the tasks $(p-d)$. The approximation is removed for those

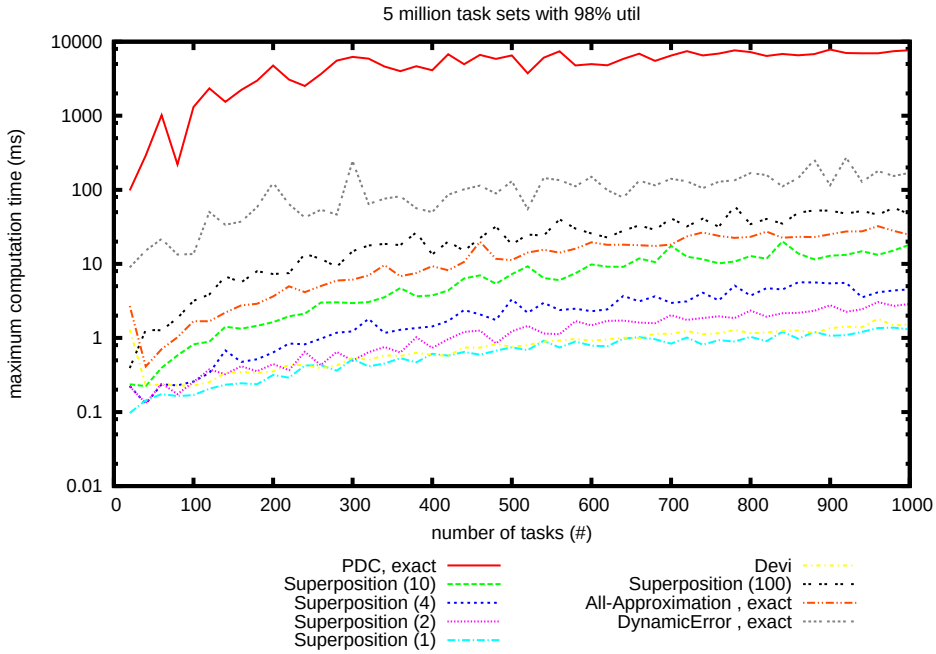

FIGURE 6.3.8. Adaptive analysis: maximum run-time of the test for different number of tasks in the task set for 98% utilization

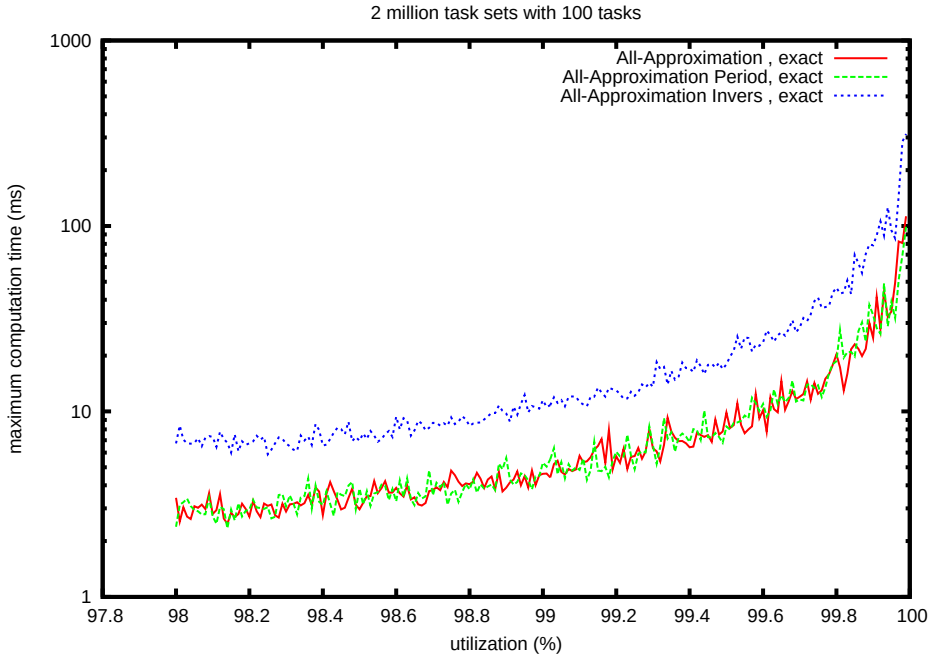

FIGURE 6.3.9. All-approximation test: different kind of orders

tasks first having the largest difference between period and deadline and therefore likely a large distance between two consecutive test intervals. Another order would be to take only the periods. In figure 6.3.9 we have compared the required maximum run-time for the

all-approximation algorithm using these two kinds of orders and, to get an impression of the influence of the kind of order, also of an all-approximation algorithm using an inverse order of the period, which means that the approximation is removed first for the task with the smallest period.

6.3.7. Conclusion. Overall the all-approximation algorithm seems to be a very good choice for nearly all applications. For online-scheduling analysis, when it is necessary to guarantee a very fast evaluation and it is possible to have task sets with very high utilizations the approximation can be a better choice.

6.3.8. Complexity of All-Approximation. In the following we will extract an estimation for the run-time complexity out of the experimental results. No formal proof for the complexity better than pseudo-polynomial exists so far. The problem for this formal proof is to find a close upper bound on the number of test intervals required by the all-approximation analysis. But calculating an estimated complexity out of the experimental data can also give hints for an upper bound on the complexity and therefore for a formal proof.

The run-time complexity for the all-approximation analysis depends on two parts. First, it depends on the run-time required (at most) for each test interval considered during the analysis and second on the maximum number of these test intervals. The run-time for each test-interval can be bounded formally by considering the algorithm 8 given in chapter 4. As we have stated there the run-time for one test-interval is bounded by $O(\log(n))$. Each test interval causes two inserting operations in priority queues, one when it is inserted into the "test-list" at a previous test interval or at the begin of the algorithm and one when it is inserted into the "approx-list". The complexity of these operations is $O(\log(n))$. The remaining operations can be bounded by a constant execution time. This is even true for the "loop" as each iteration of the loop leads to an additional new test interval and therefore, for the consideration of complexity, the effort for this loop-iteration can be assigned to this generated test interval.

So, for the estimation of complexity, it is only necessary to find an upper bound on the number of test intervals.

We will investigate the dependency of the number of test-intervals on the various parameters separately to extract the different parts of the complexity. We know that the complexity depends on the number of tasks (figure 6.3.8, table 1) in the task set and also on the utilization of the task set (figure 6.3.1, 6.3.5). In contrary to the processor demand test it does not depend on the ratio between the smallest and largest periods of tasks in a task set (figure 6.3.7).

In figure 6.3.8 the dependency of the run-time on the number of task in the task set is depicted. The run-time has to scale at least linear with the number of tasks as a test run with the result "schedulable" requires to consider at least one test interval for each task. We have done an experiment with task sets having a utilization of 98% and 20-1000 tasks (figure 6.3.8). The average numbers of test intervals for one task are for all task set sizes in the same range and only have a variation of about 8% around their medium value.

U	run-time	test points (t)	$\frac{t(1-U)}{U}$	$\frac{t\cdot(1-U)\cdot\log\left(\frac{1}{(1-U)}\right)}{U}$
98.0 %	3.488 ms	1,091	22.265	37.828
98.1 %	3.173 ms	1,112	21.537	37.070
98.2 %	3.730 ms	1,201	22.014	38.409
98.3 %	2.788 ms	1,263	21.842	38.651
98.4 %	3.180 ms	1,289	20.959	37.640
98.5 %	3.616 ms	1,458	22.203	40.496
98.6 %	3.803 ms	1,441	20.460	37.931
98.7 %	4.054 ms	1,534	20.204	38.107
98.8 %	4.215 ms	1,675	20.344	39.077
98.9 %	4.058 ms	1,810	20.131	39.430
99.0 %	4.609 ms	1,911	19.303	38.606
99.1 %	4.474 ms	2,075	18.844	38.551
99.2 %	5.225 ms	2,282	18.403	38.590
99.3 %	6.267 ms	2,533	17.856	38.478
99.4 %	5.981 ms	2,992	18.060	40.127
99.5 %	9.897 ms	3,493	17.552	40.389
99.6 %	8.413 ms	4,211	16.912	40.553
99.7 %	13.933 ms	5,305	15.963	40.272
99.8 %	20.943 ms	7,316	14.659	39.565
99.9 %	26.825 ms	14,223	14.237	42.712
99.99 %	127.80 ms	93,320	9.333	37.331

TABLE 3. All-approximation analysis: number of test-intervals in relation to some utilization dependent values

Therefore we can assume that the number of test intervals depends in the worst case linear on the number of tasks n. So we have a complexity of $O(C \cdot n \cdot \log(n))$ where C do not depends on the number of tasks.

Next we will investigate the dependency of the complexity on the utilization of the task set. Figure 6.3.1 shows the maximum run-time of the test for task sets with 100 tasks for distinguished utilizations of the task sets. The run-time is quite low for the all-approximation test for utilizations up to 80% but it increases significantly for utilizations close to 100%. We know that the complexity of the processor demand test depends on the maximum test interval and we have also proven that the last test interval considered by the all-approximation analysis is the same maximum test interval (see chapter 4). This maximum test interval (of Ripoll et al. [119], see section 2.2.3) depends on $\frac{U}{1-U}$, therefore it is valid to assume that the run-time for the all-approximation analysis also depends somehow on this value. In table 3 we have given the maximum required test-intervals of the experiment for some different utilization and the relation between these numbers of test intervals and $\frac{U}{1-U}$. In table 4 we have given the middle distance, the variation and the medium values for the columns of table 3. Let us concentrate on the values with a considerable number of test intervals, so all utilizations above 98%. We see that in the experiment the fraction $\frac{U}{1-U}$ overestimates the increase rates of the number of test intervals. The measured increase is less than the fraction would assume. Using the fraction $\frac{U}{(1-U)\cdot\log(\frac{1}{1-U})}$ we get

100 tasks per task set	$\frac{t(1-U)}{U}$	$\frac{t\cdot(1-U)\cdot\log\left(\frac{1}{(1-U)}\right)}{U}$
2 million task sets		
medium inter-distance	*11.74%*	*3.76%*
variation (%)	*41.44%*	*11.15%*
medium value	*18.498*	*36.678*
200 million task sets		
medium inter-distance	*10.88%*	*3.29%*
variation (%)	*36.98%*	*6.73%*
medium value	*19.667*	*39.230*

TABLE 4. All-approximation analysis: medium inter-distance, variation and medium values for the columns of table 3

a better estimation of the increase. The variation is about 11.15% and for 100 times more task sets it declines to 6.73%.

Our estimation achieves a complexity for the all-approximation test of

$$O\left(\frac{U}{(1-U)\cdot\log(\frac{1}{1-U})}\cdot n\cdot\log(n)\right)$$

where U is the utilization and n is the number of tasks in the task set. In comparison the processor demand criterion has a complexity of $O(\max_{\tau\in\Gamma}(p_\tau - d_\tau)\cdot n)$ when the utilization U is bounded by a value $U_{max} < 1$.

6.4. Approximation and Dynamic approximation for static priorities

In the following we will consider the run-time of the analysis algorithm for scheduling with static priorities proposed in chapter 5. We proposed the new exact exceeding cost analysis, the approximation based on it and the dynamic approximation leading to a new more efficient exact analysis. We have compared these analyses with the well known worst-case response time analysis, using the same experimental environment as in the previous sections.

Figure 6.4.1, 6.4.2 and 6.4.3 show the results of an experiment with 10 million tasks sets each having 100 tasks, a gap between 5% and 95%, a utilization between 5% and 99%. The period is chosen between 10 and 10 million ms using a normal distribution. In figure 6.4.1 the ratio of schedulable task sets is depicted. All algorithms for the analysis of static priority scheduling lead to nearly the same ratio in these figures. The ratio starts at 100% for a low utilization and drops fast at a utilization of about 80%. For comparison the figure also shows the ratio for the exact EDF scheduling calculated by the processor demand criterion (PDC), which is always larger than the ratio for static priority scheduling. In figure 6.4.2 the maximum required computation time for the different exact algorithms is shown. The most worst-case run-time required the previous worst-case response time and the new exact exceeding costs analysis. The worst-case response time requires a large run-time over all utilizations whereas the exceeding costs analysis only requires a large run-time for task sets with a high utilization (utilizations higher than 70%). For these utilizations the required run-time for the exceeding cost arbitrary analysis is often larger

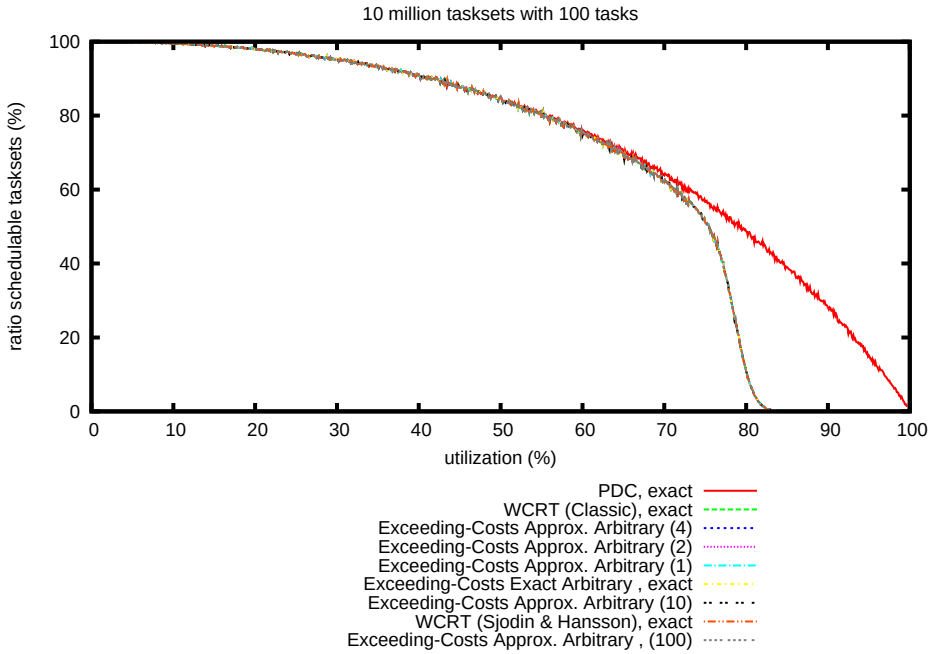

FIGURE 6.4.1. Static analysis: ratio schedulable task sets - normal distribution of periods

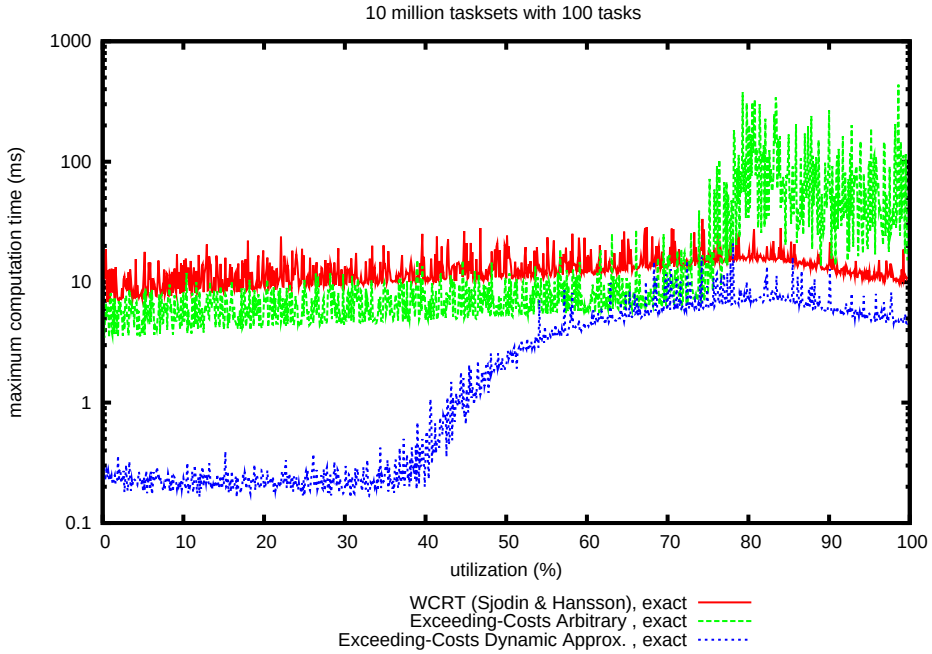

FIGURE 6.4.2. Static analysis: maximum required computation time for exact static analyses - normal distributed periods

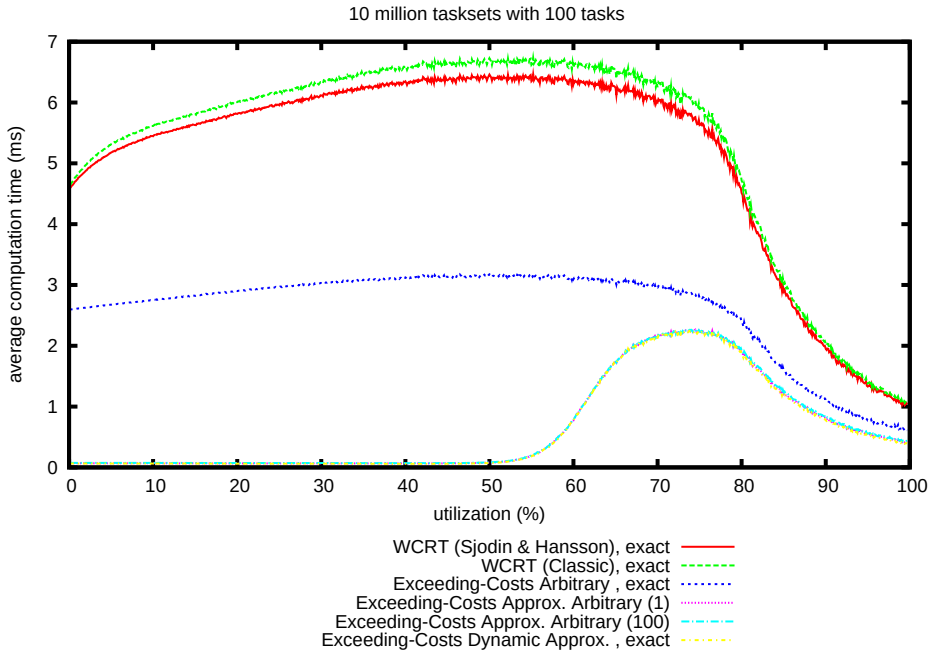

FIGURE 6.4.3. Static analysis: average run-time - normal distributed periods

than for the worst-case response-time analysis. But the approximation and the dynamic approximation of the exceeding costs analysis, which is again an exact analysis, have a better run-time than the worst-case response-time analysis. These algorithms require, in the worst-case, a run-time that is only about 20% of the run-time of the response-time analysis. It is even lower for task sets with utilizations of 70% and lower. The results are even better considering the average run-time required for task sets in the analysis as shown in figure 6.4.3.

In figure 6.4.4 the average run-time for 1 million task sets with 500 tasks each is depicted. In figure 6.4.5 the dependency of the computation time on the number of tasks in the task set is depicted. The interesting point is that the effort even for the exact non-adaptable static exceeding cost analysis drops below the effort for the worst-case response-time analysis for large task sets.

We focus on exponentially distributed periods next. Figure 6.4.6 shows the run-times of the same experiment with exponential distribution for the periods. At a utilization of 40% the top curve in the figure is the exceeding-costs arbitrary analysis, followed by the variation of the worst-case response time analysis, which leads to equal results and cannot be distinguished in the figure. The next lower curve at 40% utilization is the exceeding costs approximation with an approximation degree of 100 exact test intervals followed by the approximation with one exact test interval. The lowest curve is the exact dynamic exceeding-cost approximation. The advantage of the new approximation and dynamic-approximation is even higher than in the previous case. The exponential distributions lead to task sets requiring more effort for the response-time analysis but not for the dynamic-approximated exceeding-cost analysis. The advantage is again even higher considering

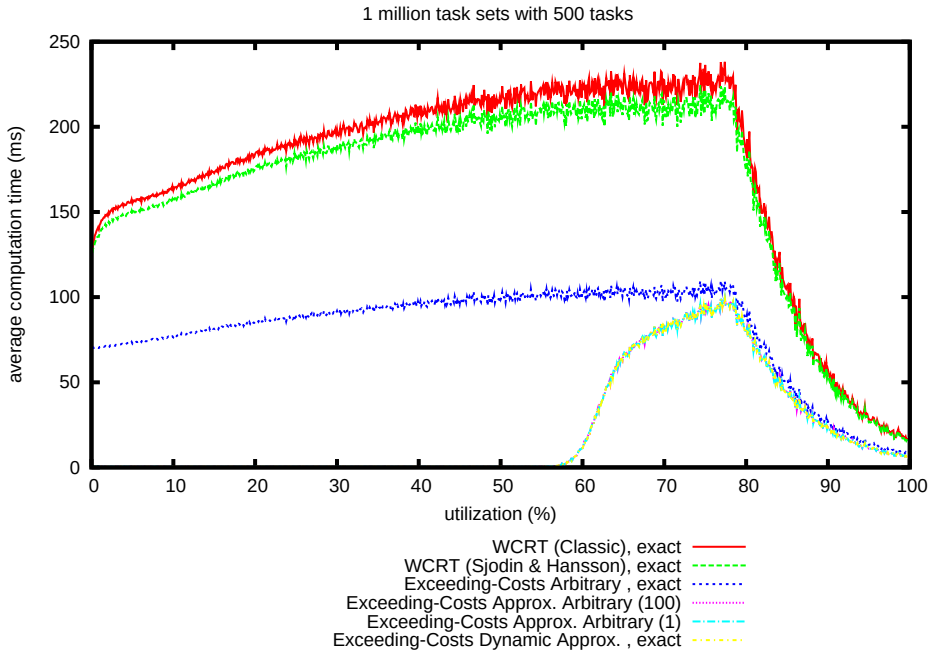

FIGURE 6.4.4. Static analysis: average run-time - normal distributed periods (500 tasks)

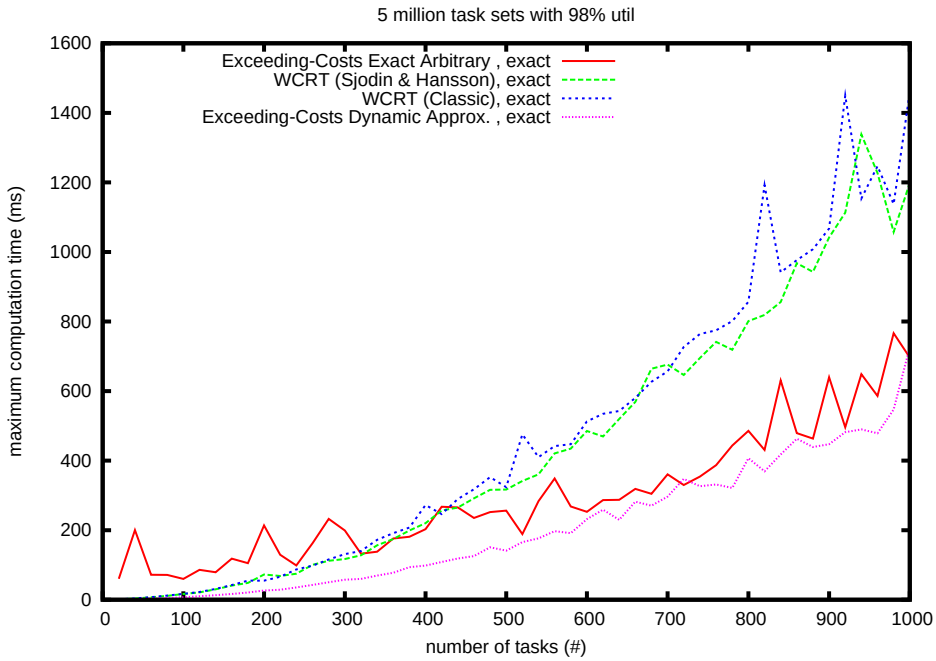

FIGURE 6.4.5. Static analysis: maximum run-time for different number of tasks - normal distributed periods

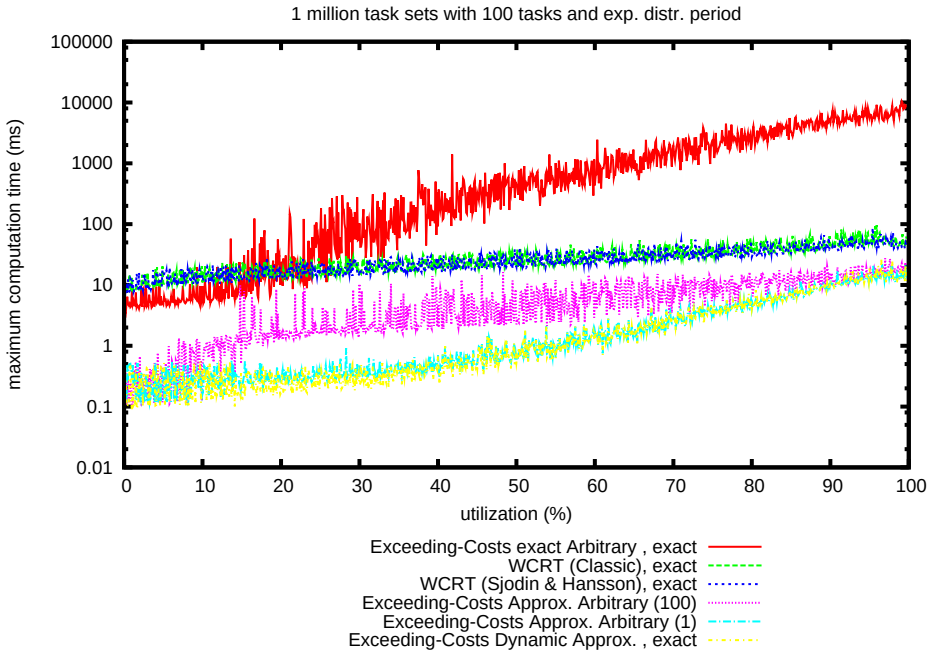

FIGURE 6.4.6. Static analysis: maximum required run-time for approximative static analyses algorithms - exponential distributed periods

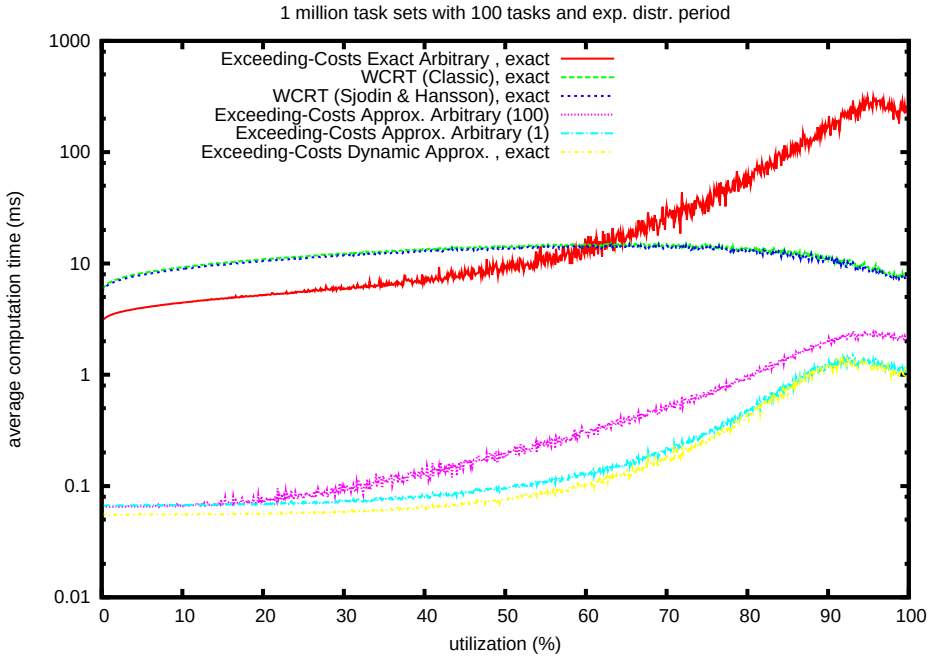

FIGURE 6.4.7. Static analysis: average run-time - exponential distributed periods

the average required run-time for the whole experiment as shown in figure 6.4.7. For the utilization of 0% the highest curve is here the result of both worst-case response time

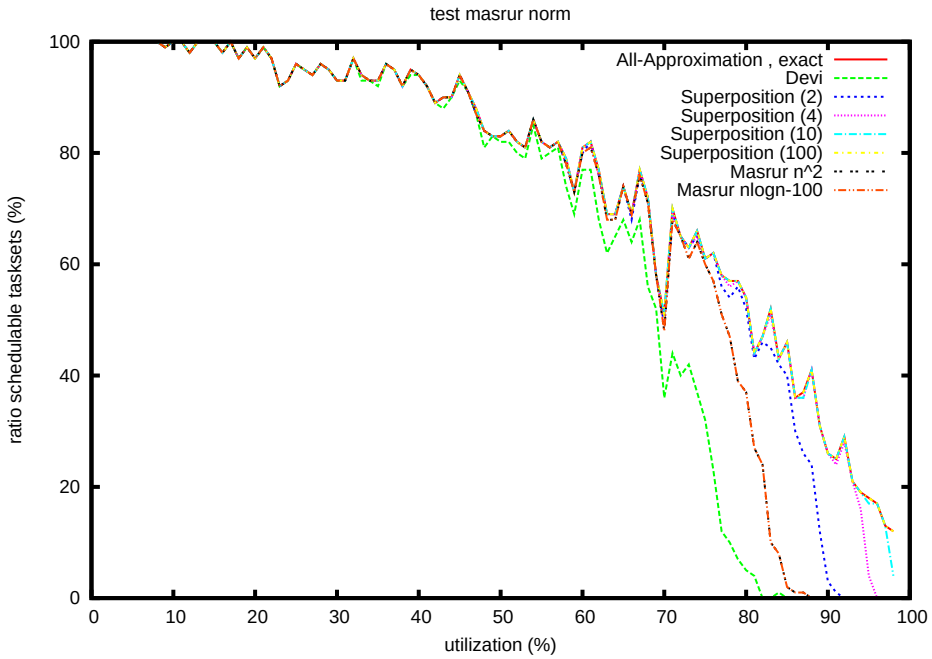

FIGURE 6.5.1. EDF: acceptance ratio of the approach of Masrur et al. (normal distribution)

analyses followed by the arbitrary exceeding-costs analysis. The lower curves are both approximations (100 and 1 exactly considered test interval) and the lowest curve is again the dynamic exceeding-cost analysis. In contrary to the processor demand criterion the ratio between the smallest and largest task of a task set has no large influence on the run-time of the response-time analysis.

Although results are not as convincing as the results of the analysis for dynamic priorities the new algorithm is still an improvement over the existing approaches.

6.5. Previous approaches

Finally we will consider the run-time of the other exact analysis approaches introduced in chapter 2 and compare them with our approach.

6.5.1. Analysis of Masrur et al. [93].

In figure 6.5.1 and figure 6.5.2 we have depicted the acceptance ratio and the maximum required run-time for the sufficient test of Masrur et al. [93] for EDF scheduling and compared them with the other approaches. Both, the acceptance ratio and the effort is located between the sufficient test of Devi which is equal to the approximation with one considered test interval and the approximation with two considered test intervals. Using an exponential distribution the sufficient test performs better (figure 6.5.3 and figure 6.5.4).

6.5.2. Exact analysis algorithms for EDF. In figure 6.5.5 the average and in figure 6.5.6 the maximum run-time for the different schedulability analyses and maximum test intervals for EDF scheduling are compared. In both cases the processor demand test with

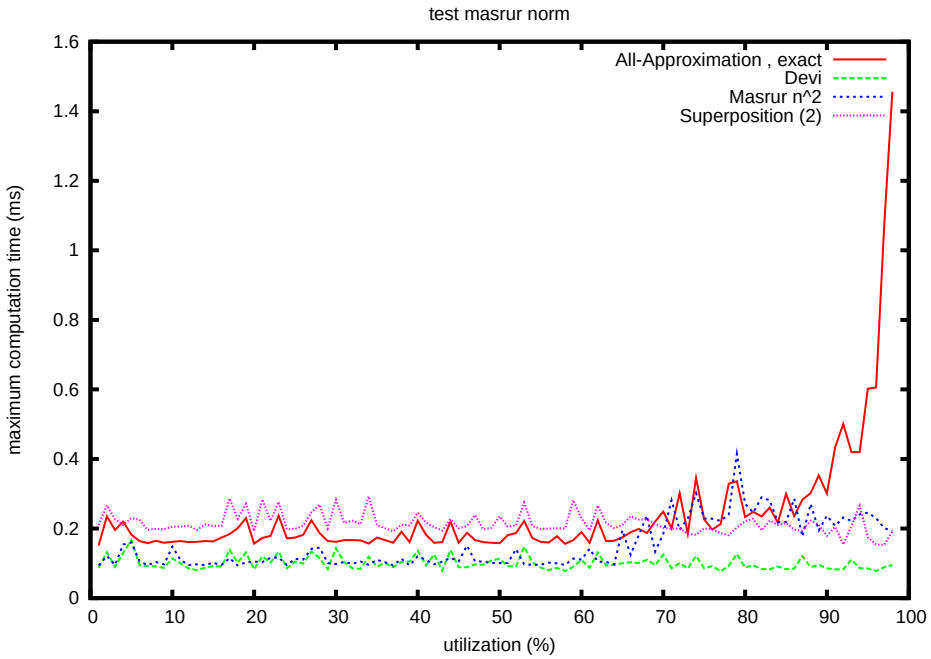

FIGURE 6.5.2. EDF: max run-time compared of approach of Masrur et al. (normal distribution)

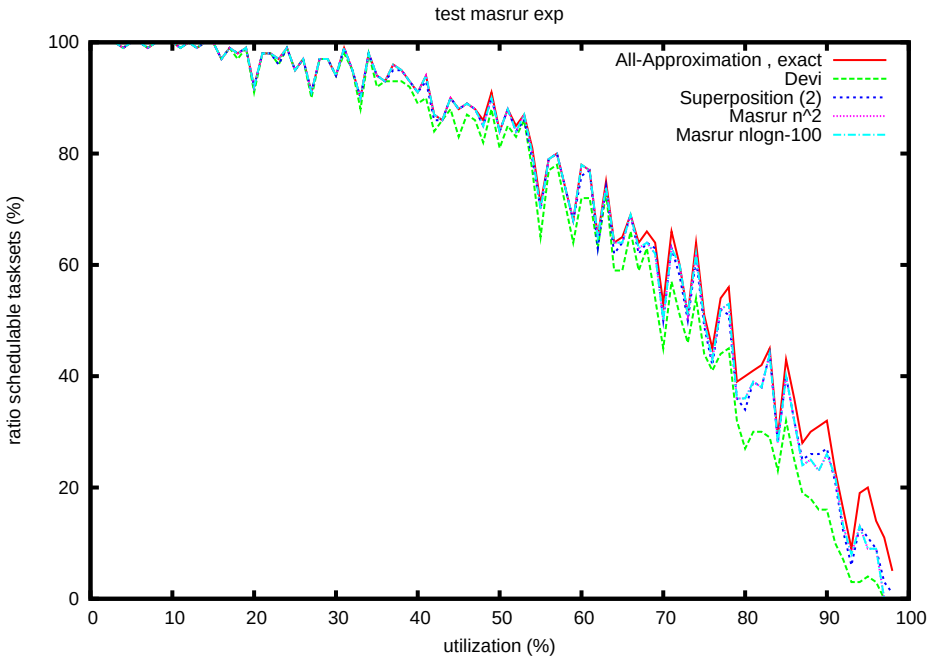

FIGURE 6.5.3. EDF: acceptance ratio of the approach of Masrur et al. (exp. distribution)

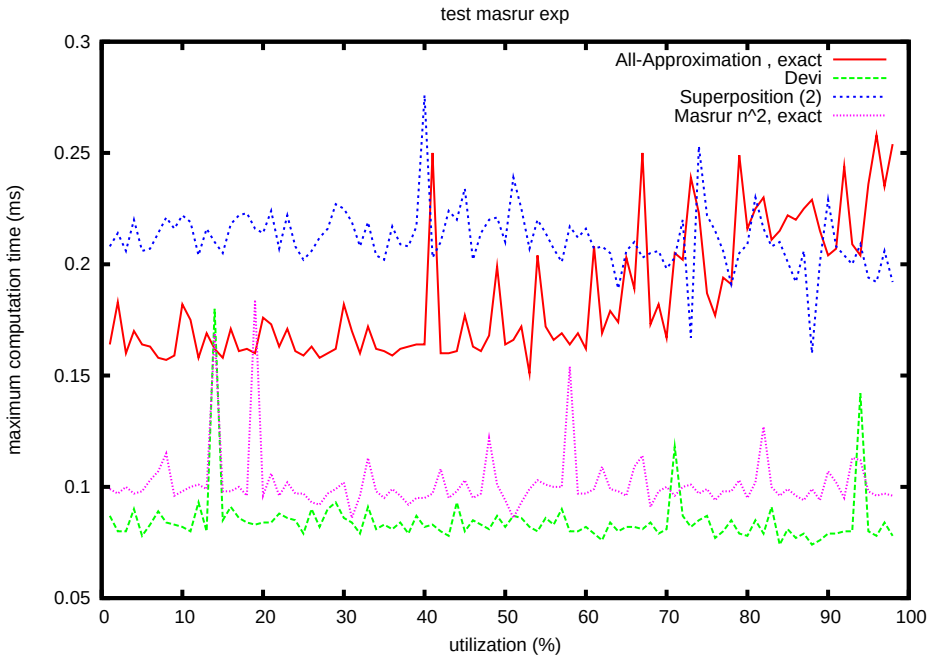

FIGURE 6.5.4. EDF: max run-time compared of approach of Masrur et al. (exp. distribution)

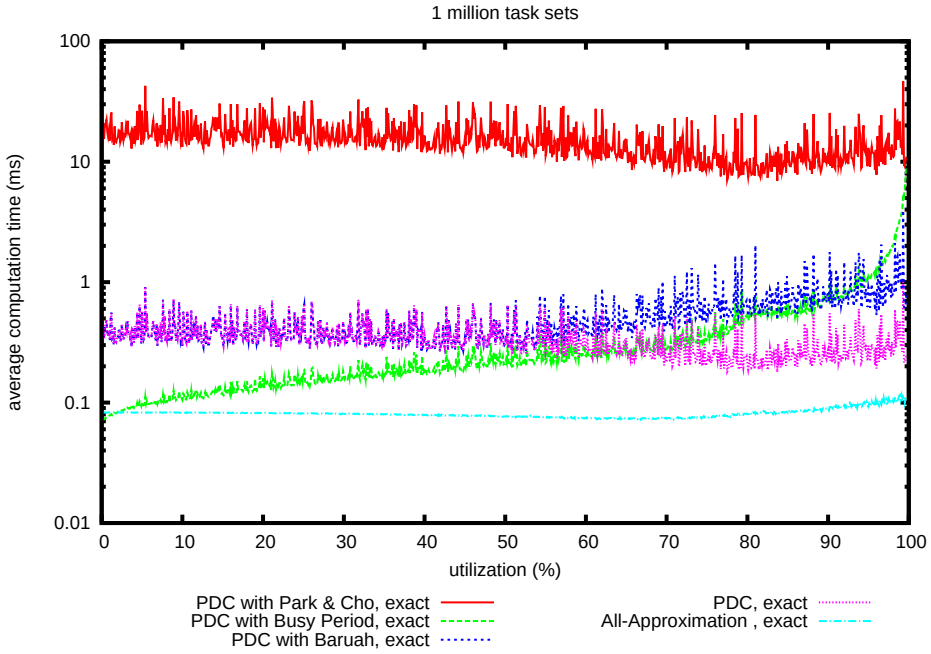

FIGURE 6.5.5. EDF: average run-time of the previous approaches

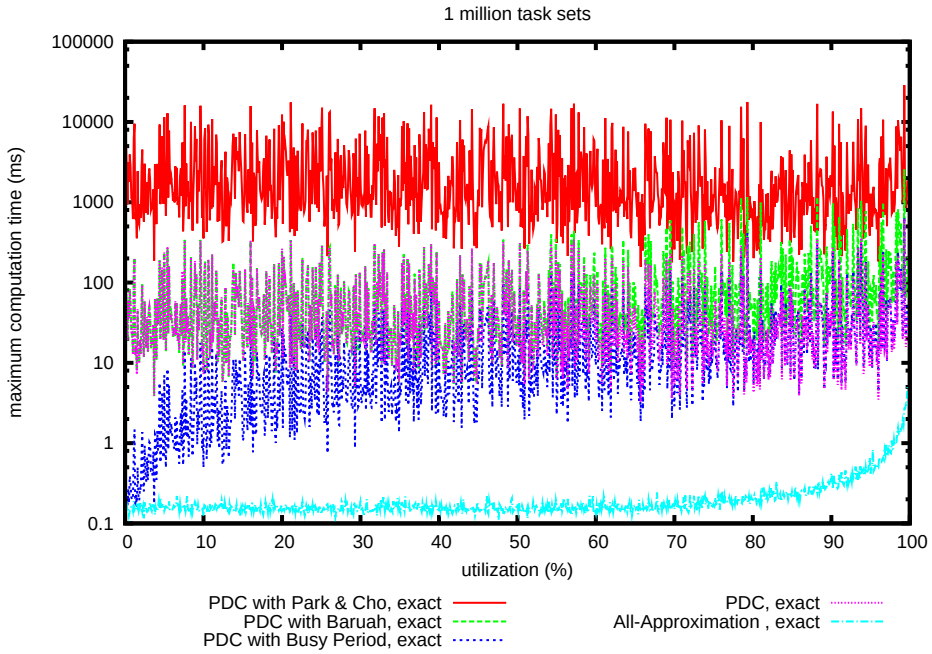

FIGURE 6.5.6. EDF: maximum run-time of the previous approaches

the test interval of Park & Cho [103] requires the most effort. The reason is that the test requires a recalculation of its maximum test interval at each test interval. In the average the PDC with the maximum test interval of Baruah et al. [19] requires the same effort as the PDC with the test bounds of Ripoll et al. [119] or George et al. [55] (shown as "PDC, exact" in the figure and used in the previous sections) for low utilizations but it requires significantly more effort for utilizations of more than 50% as it leads to longer maximum test intervals. The PDC with the busy period as maximum test interval ([119]) has for very low utilizations an average run-time comparable with the run-time of all-approximation but requires with rising utilization a higher run-time. For utilizations close to 100% the average required run-time is even comparable to the run-time of the test of Park & Cho [103]. The all-approximation analysis requires the least effort of all approaches.

For the maximum run-time the situation is comparable to the average case except that all previous approaches have a large fluctuation in their required run-times.

6.5.3. Exact analysis algorithms for static priority scheduling. In figure 6.5.7 and figure 6.5.8 we have compared the average and the maximum required run-time for exact schedulability analysis for static priorities. The scheduling-point test requires the maximum runtime for the relevant utilizations between 75% and 100% followed by the reduced scheduling-point test. The worst-case response-time analysis is the next with a required maximum runtime of around 10 ms followed by the exceeding-cost dynamic approximation. But for utilizations between 50% and 75% the situation is a little different as here the scheduling-point test and the reduced scheduling-point test often require even less runtime than the exceeding-cost dynamic approximation. The situation is different for the

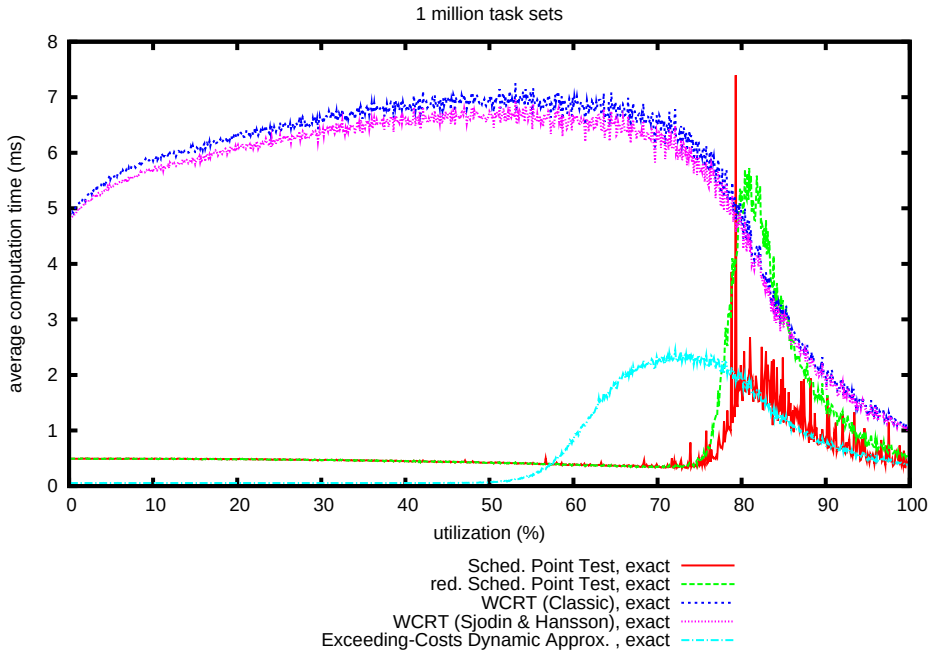

FIGURE 6.5.7. Static priorities: average run-time of previous approaches

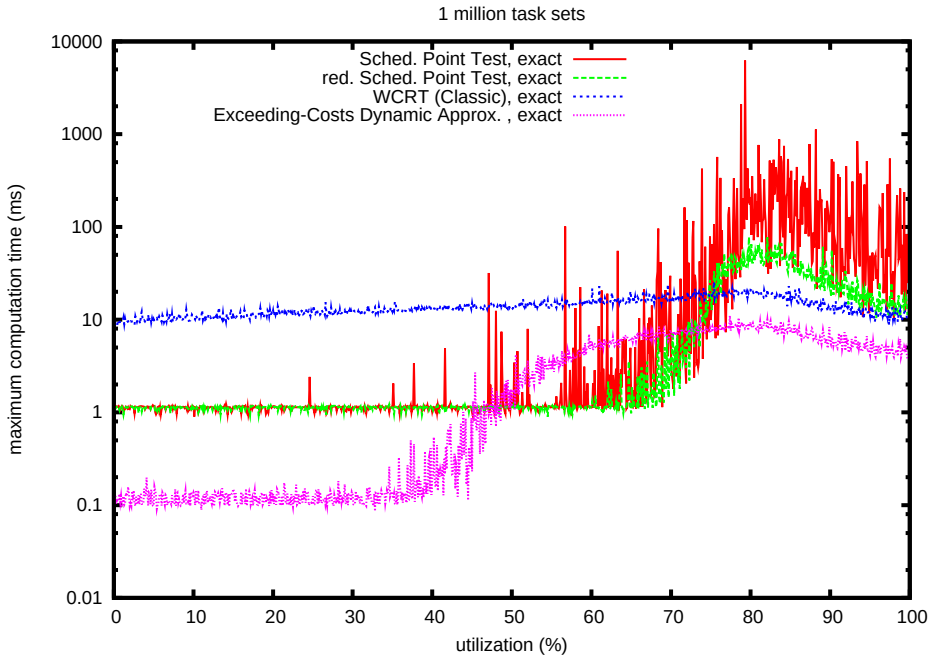

FIGURE 6.5.8. Static priorities: maximum runtime of previous approaches

average runtime. Here the runtime for the reduced scheduling-point test is, with some exceptions, even for utilizations larger than 80% comparable with the exceeding-costs dynamic approximation, but it has a larger fluctuation. The effort for the scheduling-point

test increases suddenly at about 78% utilization and exceeds even the effort for the worst-case response-time analysis there. Note, that the average case of course includes many uninteresting task sets far away from the border between schedulable and non-schedulable task sets. Concluding, the effort for the scheduling-point test and the reduced scheduling-point test seems to be unpredictable due to their large fluctuation and the exceeding-costs dynamic approximation seems to be the best choice.

CHAPTER 7

Hierarchical event spectra

Sub-additive and super-additive event bound functions (and also service bound - capacity bound functions) are a key concept for the schedulability analysis. The answer of the question how many events can occur at most - at least within any possible interval of length Δt leads to a integrated theory on schedulability analysis. These functions extract the worst-case situations of all possible concrete schedules into one single description. We call this concept event spectrum, because an event spectrum contains all possible worst-case event densities like the light spectrum contains all possible wavelengths of the different colors of light. An efficient and compact description for event spectra, allowing a fast calculation of the values for each interval, leads to an efficient real-time analysis. Many proposed models in the real-time community are, in reality, concrete descriptions of event spectra or are used in the same way. Examples are the event stream model [61] (introduced in section 2.3.2), the periodic model with jitter and minimum separation distance [117] (introduced in section 2.3.4), the concrete description and the approximation of the real-time calculus curves [39, 76] (introduced in section 2.3.6). But also most of the analysis proposed for the periodic model, the periodic model with jitter and the recurring real-time task model are based somehow on event spectra. Of course, the approximation and the analysis algorithm proposed for these models in chapter 3, chapter 4 and chapter 5 are also based on event spectra. One possible description model for event spectra is the event stream model; another is the periodic model with jitter and minimum separation distance between events.

In this chapter we will present a new advanced concrete description model to overcome the limitations of the event stream model, the hierarchical event spectra model. The model allows the efficient concrete description of event spectra containing various kinds of bursts and capacity bound functions.

7.1. Limitations of the event stream model

The event stream model is a general event model. In principle an event stream can describe each possible event pattern. But for the description of certain event patterns by an event stream a large number of event elements are required. Consider for example an event pattern in which 100 events occurs with a period of 10 ms followed by a break of 1010 ms. After the break the next 100 events arrived followed by the break and so on. The description of this pattern with a periodic event sequence would be:

$$\Theta = \{(2000\,ms, 0), (2000\,ms, 10), (2000\,ms, 20), (2000\,ms, 30), ..., (2000\,ms, 990)\}$$

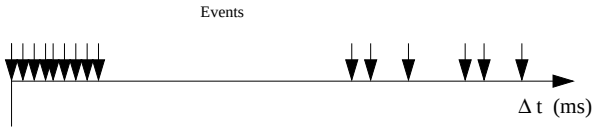

FIGURE 7.1.1. Example Event Spectrum

The description of such a pattern with a periodic event sequence requires 100 event elements. As the event pattern has a very regular appearance it should be possible to describe the pattern in a more compact way.

Bursts can be found in the activating event patterns for certain systems. They occur also within a system itself.

EXAMPLE 7.1.1. Consider for example a robot with a camera sending frequently pictures to a processor using a bus. Every time the camera sends a picture, a burst occurs on the bus. The burst consists of the data-packages with the information for one picture. After the bursts a delay occurs on the bus until the next picture is send. Note, that the single data packages of a picture are not necessarily of the same size or need not to occur with a strict periodic pattern. They can contain quite different information, picture data as well as protocol or header information and it might be useful or necessary to describe the bursts itself also by an event pattern.

A burst is a large but limited number of events occurring within a short amount of time. It can be regarded as a limited event pattern having a high density of events. Modeling bursts with the event stream model can require a large number of event elements.

EXAMPLE 7.1.2. An example for a system in which bursts occur is a resource ρ with a task set Γ bound on it for which the completion of τ can jitter due to delays caused by preempting tasks $\tau' \in \Gamma$ with higher priorities $(\tau' \in hp(\tau))$. In case of a large jitter value j several jobs of τ can be delayed at the same point of time. These jobs can be processed and finished quickly in a row after the execution of the jobs of the preempting tasks has been completed. The outgoing events fired by these jobs than occur one after another also within a short amount of time, forming a burst. This example was the motivation for the development of the periodic event model with bursts by Richter et al. [117], which is introduced in section 2.3.4. But the periodic model with bursts cannot describe efficiently the event patterns for the robot with a camera example.

EXAMPLE 7.1.3. Bursts can also be the result of loops in the control flow graph of previous tasks. Consider a task having a control flow with a loop in its control-flow in which one event is generated at each of the iterations of the loop. In case, that the number of iterations of the loop can be bounded, the number of events occurring out of one activation of the task is bounded too. The set of events generated by a single activation of the loop can be regarded as a burst.

In Figure 7.1.2 we present an example task graph to illustrate bursts. τ_1 is activated by a periodic event stream. It activates two other task τ_2 and τ_3. The event stream for both tasks consists mainly of bursts. The control flow graph of τ_1 consists of two nested loops.

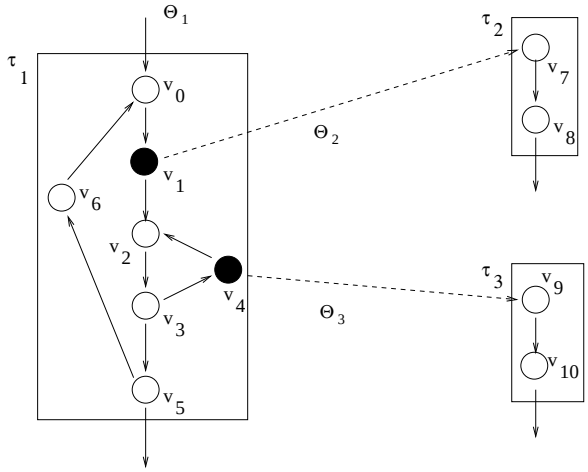

FIGURE 7.1.2. Example task graph generating bursts

In the outer loop node v_1 generates the events activating τ_2. v_4 is the node belonging to the inner loop that generates the events activating τ_3. One event is generated at each of the iterations of the inner loop. Therefore the event stream Θ_2 activating τ_2 mainly consists of bursts, whereas Θ_3 activating τ_3 consists of nested bursts due to the nested loops. A new model is required that is able to describe such event patterns in an easy and efficient way and allows also an efficient real-time analysis.

The event stream model is not appropriate to model efficiently the systems with bursts introduced in the examples. The problem is that it is necessary to model each event in the burst with an extra event stream element. For an event pattern containing a burst with 10 events, at least 10 event stream elements are required. In case of a burst containing 1000 events, at least 1000 event stream elements are necessary. The size of the description for an event pattern depends in the event stream model on the size of its bursts.

Another limitation of the event stream model becomes visible due to the proposed approximations. It is not possible to describe the approximation of event stream elements within the event stream model. Therefore it was necessary to develop a special approximative feasibility test algorithm, implementing a fixed test limit for all tasks. It would be more useful to integrate the approximation in the model to describe the event stream with both, its exact and its approximated part. The new model would allow a higher flexibility in choosing the test limits for the individual event stream elements and give the freedom to propagate the approximation through a distributed system.

In the following we will propose such a model: The hierarchical event spectra model.

7.2. Spectra

Looking to the previous work to analyze real-time systems shows that the analysis complexity and the model accuracy are central points of the ongoing research. Because of the limitations of the models a lot of extensions to cover jitter, burst and other effects of events are discussed in the literature. The new hierarchical spectra or hierarchical event

spectra model presented in this thesis will cover all aspects in a new single model. It is based on event streams and their approximations [5]. Event spectra are a hierarchical extension of the event stream model (previously referred to as hierarchical event streams) combined with results from the real-time calculus. The new model is called event spectra, because an event spectrum contains all possible worst-case event densities like the light spectrum contains all possible wavelengths of the different colors of light.

We will first give a definition for the general spectrum model, which can cover many aspects of a system. A spectrum can model for example the number of events, the amount of required computation time, the amount of available capacity or the amount of required or available energy, in relation to time intervals. A spectrum describing events is denoted event spectrum, when it is describing costs it is denoted cost spectrum and it is denoted capacity spectrum when it describes the (available or remaining) capacity. To describe many different system stimuli the event spectrum has, for example, to model a sequence of events in an accurate and complete way.

DEFINITION 7.2.1. *Spectrum*

Let a spectrum $\hat{\Theta}$ model the relation between an amount of a parameter (like number of events, required computation time, available capacity, required energy, ...) and a time-interval length Δt. Let the hierarchical spectrum $\hat{\Theta} = \{\hat{\theta}\}$ consist of a set of hierarchical spectrum elements. Let the hierarchical spectrum element $\hat{\theta} = (p_{\hat{\theta}}, a_{\hat{\theta}}, L_{\hat{\theta}}, f_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}})$ be described by a period $p_{\hat{\theta}}$, an offset $a_{\hat{\theta}}$ and a limitation $L_{\hat{\theta}}$ of the amount generated within one single period. The slope $f_{\hat{\theta}}$ is describing a constantly growing amount with $f_{\hat{\theta}}$ as the growing rate. $\hat{\Theta}_{\hat{\theta}}$ is a child hierarchical spectrum which is recursively embedded.

The amount of the parameter that can be generated by $\hat{\Theta}_{\hat{\theta}}$ within one period of its parent spectrum $\hat{\theta}$ is also bounded by the limitation of its parent $L_{\hat{\theta}}$. $\hat{\Theta}_{\hat{\theta}}$ can be an empty element ($\hat{\Theta}_{\hat{\theta}} = \emptyset$).

$\hat{\theta}$ is a valid hierarchical spectrum element, if and only if $\hat{\theta}_{\hat{\theta}}$ is either a valid hierarchical spectrum or an empty element and the interval Δt required for the slope and $\hat{\Theta}_{\hat{\theta}}$ to generate the amount of the limitation $L_{\hat{\theta}}$ is not larger than the period $p_{\hat{\theta}}$ (Separation Condition, see condition 7.2.6 for a mathematical definition) and either $f_{\hat{\theta}} = 0$ or $\hat{\Theta}_{\hat{\theta}} = \emptyset$.

The event bound function (as defined in definition 2.3.3) calculates the maximum number of events that can be generated by a given event spectrum $\hat{\Theta}$ within an interval of a given length Δt. The spectrum bound function $\eta(\Delta t, \hat{\Theta})$ has the same definition as the event bound function of the event stream model. The separation condition allows an efficient calculation of the values for this function.

LEMMA 7.2.2. *Spectrum Bound Function:*

Let for any $\Delta t, p$ define $mod(\Delta t, p) = \Delta t - \left\lfloor \frac{\Delta t}{p} \right\rfloor p$. For a hierarchical spectrum fulfilling the separation condition the spectrum bound function can be determined as follows:

$$\eta(\Delta t, \hat{\Theta}) = \sum_{\substack{\hat{\theta} \in \hat{\Theta} \\ \Delta t \geq a_{\hat{\theta}}}} \eta(\Delta t, \hat{\theta})$$

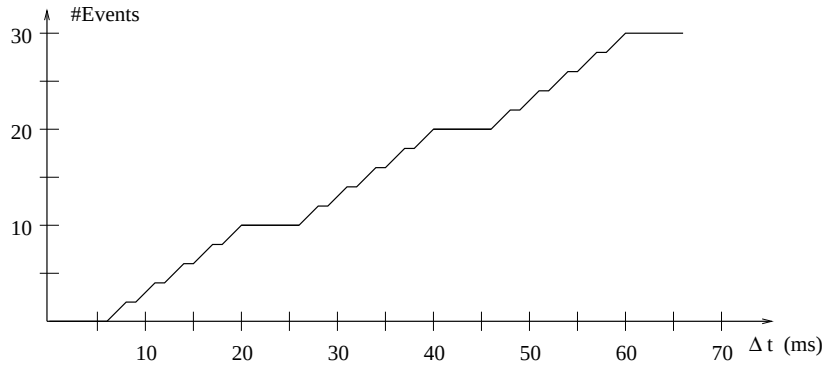

FIGURE 7.2.1. Hierarchical event spectrum $\hat{\Theta}_6$

$$\eta(\Delta t,\hat{\theta}) = \begin{cases} L_{\hat{\theta}} & p_{\hat{\theta}} = \infty, f_{\hat{\theta}} = \infty \\ min(L_{\hat{\theta}}, (\Delta t - a_{\hat{\theta}}) f_{\hat{\theta}} + \eta(\Delta t - a_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}})) & p_{\hat{\theta}} = \infty, f_{\hat{\theta}} \neq \infty \\ \left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} + 1 \right\rfloor L_{\hat{\theta}} & p_{\hat{\theta}} \neq \infty, f_{\hat{\theta}} = \infty \\ \left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rfloor L_{\hat{\theta}} + min(L_{\hat{\theta}}, mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}) f_{\hat{\theta}} + & \\ \qquad + \eta(mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}})) & p_{\hat{\theta}} \neq \infty, f_{\hat{\theta}} \neq \infty \end{cases}$$

with

$$\eta(\Delta t, \emptyset) = 0$$

PROOF. Due to the separation condition it is always possible to include the maximum allowed amount for completed periods $\left(\left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rfloor L_{\hat{\theta}}\right)$. Only the last incomplete fraction of a period has to be considered separately. This remaining interval is given by subtracting all completed periods, and the offset a from the interval Δt $\left(mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}\right)$. It has to be distinguished whether the slope or the sub-spectrum generates the events. In case of the sub-spectrum, the possible amount is calculated by using the same spectrum bound function with the remaining interval and the new embedded hierarchical event spectrum as parameters. In case of the slope the amount is simply the product of the slope f and the remaining interval length $(mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}))$. The maximum allowed amount within one period limits both values. □

Independently on which hierarchical level a hierarchical spectrum element is located it is visited only once during the calculation of the value for one interval. This characteristic limits the complexity of the calculation.

It is not necessary for the spectra to be homogeneous.

EXAMPLE 7.2.3. Let us consider the hierarchical spectrum

$$\hat{\Theta}_6 = \{(20\,ms, 6\,ms, 10, 0\frac{1}{s}, \{(3\,ms, 0\,ms, 2, 1\frac{1}{s}, \emptyset)\}$$

The event bound function for this spectrum is given in figure 7.2.1. The value for $\Delta t = 33$ ms is given by

$$\eta(33\,ms, \hat{\Theta}_6) = \left\lfloor \frac{27\,ms}{p_{\hat{\theta}}} \right\rfloor L_{\hat{\theta}} + \min(L_{\hat{\theta}}, mod(27\,ms, p_{\hat{\theta}}) f_{\hat{\theta}} + \eta(mod(27\,ms, p_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}}))$$

$$= \left\lfloor \frac{27\,ms}{20\,ms} \right\rfloor \cdot 10 + \min(10, 0 + \eta(7\,ms, \hat{\Theta}_{\hat{\theta}})) = 10 + \min(10, \eta(7\,ms, \hat{\Theta}_{\hat{\theta}}))$$

$$\eta(7\,ms, \hat{\Theta}_{\hat{\theta}}) = \eta(7\,ms, \hat{\theta}') = \left\lfloor \frac{7\,ms}{3\,ms} \right\rfloor \cdot 2 + min(2, mod(7,3) \cdot 1 + 0) = 4 + 1 = 5$$

$$\eta(33\,ms, \hat{\Theta}_6) = 10 + \min(10, 5) = 15$$

The spectrum bound function $\eta(\Delta t, \hat{\Theta})$ allows calculating the amount for a given interval-length Δt. It provides for each interval-length Δt the amount belonging to Δt. It is a monotonic rising function $(\Delta t_i > \Delta t_j \rightarrow \eta(\Delta t_i) \geq \eta(\Delta t_j)$. An interval Δt_i has to include at least all the amount of Δt_j because in the worst-case Δt_j can be part of Δt_i.

The spectrum bound function allows to calculate the amount for a given interval-length.

DEFINITION 7.2.4. *Interval bound function*

The interval bound function ψ calculates the time interval Δt for a given event spectrum and a given number of events.

$$\psi(x, \hat{\Theta}) = min(\Delta t | x = \eta(\Delta t, \hat{\Theta}))$$

LEMMA 7.2.5. *ψ is the inverse of the spectrum bound function so we have $\eta(\psi(x, \hat{\Theta}), \hat{\Theta}) = x$ and $\psi(\eta(\Delta t, \hat{\Theta}), \hat{\Theta}) \leq \Delta t$.*

PROOF. Let us assume, without loosing the generality, $\Delta t'$ being the interval fulfilling $\Delta t' = min(\Delta t | x = \eta(\Delta t, \hat{\Theta}))$. Therefore $\eta(\psi(x, \hat{\Theta}), \hat{\Theta}) = \eta(\Delta t', \hat{\Theta}) = x$ and $\psi(\eta(\Delta t, \hat{\Theta}), \hat{\Theta}) = \psi(x, \hat{\Theta}) = \Delta t' \leq \Delta t$. □

The separation condition prohibits that the amounts of different periods of a spectrum element overlaps. The separation condition is required for an efficient calculation of the spectrum bound function. It can be mathematically expressed as

CONDITION 7.2.6. *(Separation Condition) A spectrum element $\hat{\theta}$ fulfills the separation condition if for each element the interval in which events are generated is equal or smaller than its period:*

$$\eta(L_{\hat{\theta}}, \hat{\theta}) + \frac{L_{\hat{\theta}}}{f_{\hat{\theta}}} \leq p_{\hat{\theta}}$$

or

$$p_{\hat{\theta}} \leq \eta(p_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}}) + \frac{p_{\hat{\theta}}}{L_{\hat{\theta}}}$$

The condition 7.2.6 does not reduce the space of event patterns that can be modeled by an event spectrum. An event spectrum that does not meet the separation condition can easily be transferred to one meeting this condition.

COROLLARY 7.2.7. *A spectrum element $\hat{\theta}$ that does not meet the separation condition can be exchanged by a set of spectrum elements $\hat{\theta}_1, ..., \hat{\theta}_k$ with $k = \left\lceil \frac{\psi(L_{\hat{\theta}}, \hat{\theta})}{p_{\hat{\theta}}} \right\rceil$ and $\hat{\theta}_i = (kp_{\hat{\theta}}, (i-1)p_{\hat{\theta}} + a_{\hat{\theta}}, L_{\hat{\theta}}, f_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}})$.*

PROOF. For any spectrum element we can find a multiple of its period $kp_{\hat{\theta}}$ that is larger than the interval in which events are generated by this element. For each period of

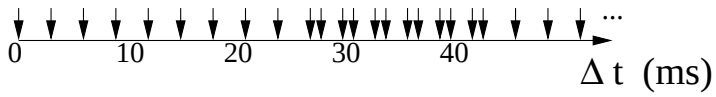

FIGURE 7.2.2. Example for overlapping events of different periods

the original spectrum element, a seperate spectrum element is used with the period $kp_{\hat{\theta}}$ and the events of the original element. Each of these new elements fulfills the separation condition. □

EXAMPLE 7.2.8. Consider the following example event spectrum:

$$\hat{\theta} = \{(28ms, 0ms, 15, 0\frac{1}{s}, \{(3ms, 0ms, 1, \infty\frac{1}{s}, \emptyset)\})\}$$

The limitation interval $L_{\hat{\theta}}$ has the length $L_{\hat{\theta}} = (15-1)\cdot 3ms = 42ms$. The event pattern of this event spectrum is shown in figure 7.2.2. The first and the second period of the event spectrum element overlap. In the interval $[0,42]$ ms the events for the first period are generated, in the interval $[28,70]$ ms the events of the second period. Both intervals are overlapping in the interval $[28,42]$ ms in which events of both periods are generated. $\hat{\Theta}$ can be transferred into the following event spectrum $\hat{\Theta}'$ meeting the separation condition:

$$\begin{aligned}\hat{\Theta}' \quad = \quad & \{(56ms, 0ms, 15, 0\frac{1}{s}, \{(3ms, 0ms, 1, \infty\frac{1}{s}, \emptyset)\}),\\ & (56ms, 28ms, 15, 0\frac{1}{s}, \{(3ms, 0ms, 1, \infty\frac{1}{s}, \emptyset)\})\}\end{aligned}$$

Note that this separation condition does not prevent the intervals for event generation of different spectrum elements to overlap. It is only not allowed that intervals for event generation of two periods of the same spectrum element can overlap.

DEFINITION 7.2.9. *Upper spectrum* $\hat{\Theta}^+$

$\hat{\Theta}$ is an upper spectrum $\hat{\Theta}^+$ if, and only if, each of its spectrum elements are valid spectrum description elements and the condition of sub-additivity

$\forall \Delta t, \Delta t' : \eta(\Delta t + \Delta t', \hat{\Theta}^+) \leq \eta(\Delta t, \hat{\Theta}^+) + \eta(\Delta t', \hat{\Theta}^+)$ is fulfilled.

DEFINITION 7.2.10. *Lower spectrum* $\hat{\Theta}^-$

$\hat{\Theta}$ is a lower spectrum $\hat{\Theta}^-$ if, and only if, each of its spectrum elements are valid spectrum description elements and the condition of super-additivity

$\forall \Delta t, \Delta t' : \eta(\Delta t + \Delta t', \hat{\Theta}^-) \geq \eta(\Delta t, \hat{\Theta}^-) + \eta(\Delta t', \hat{\Theta}^-)$ is fulfilled.

The spectrum $\hat{\Theta}^+$ models the maximum amount that can occur in an interval Δt and $\hat{\Theta}^-$ models the minimum amount that can occur within Δt.

A value $f_{\hat{\theta}} = 1$ of the slope means that after one time unit one event has occurred, after two time units two events and so on. The slope allows modeling approximated event streams as well as modeling the capacity of resources. Both cases can be described by a number of events which occur respectively can be processed within one time unit.

EXAMPLE 7.2.11. An event spectrum $\Theta_1 = \{(10ms, 5ms, 1, \infty, \emptyset)\}$ with one element with an offset $a_\theta = 10$ ms and a period $p_\theta = 5$ ms can be approximated by a slope $f = \frac{1}{5ms}$.

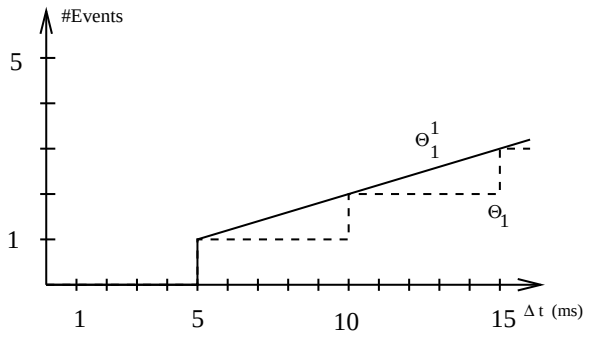

FIGURE 7.2.3. Example event spectrum

In case that this event pattern is approximated after the first event it can be described by the approximated event spectrum

$$\hat{\Theta}_1 = \{(\infty s, 10\,ms, 1, \infty \frac{1}{s}, \emptyset), (\infty s, 10\,ms, \infty, \frac{1}{5\,ms}, \emptyset)\}$$

The approximated pattern is visualized in figure 7.2.3. As the original event spectrum produces an unlimited number of events the approximated event spectrum is not bounded, too and has therefore an infinite limitation ($L_{\hat{\theta}_{1.1}} = \infty$) . This unlimited slope is not repeated periodically, therefore $p_{\hat{\theta}_{1,1}} = \infty$. To guarantee that $\hat{\theta}_1$ leads to an equal or larger amount of events for each interval than Θ_1 the approximation has to start at the right level guaranteed by the first element of $\hat{\Theta}_1$. Then $\hat{\Theta}_1$ leads to a value of two events when the second event of Θ_1 occurs at 15 ms.

EXAMPLE 7.2.12. One event which occurs immediately requires an infinite slope ($f_{\hat{\theta}} = \infty$) and a limitation of one ($L_{\hat{\theta}} = 1$). It can be described by the following hierarchical event spectrum element:

$$\mathrm{e} \quad = \quad \{(\infty s, 0\,s, 1, \infty \frac{1}{s}, \emptyset)\}$$

A recursively embedded event spectrum with a slope of $f = \infty$ would lead to the generation of an infinite number of events in no time, but due to the limitation only the generation of one event is possible. Due to the offset zero the generation of events can start immediately, therefore *e* generates one single event at time zero.

EXAMPLE 7.2.13. A capacity function of a resource which can handle one second processing time in one second can be described by this model with one capacity spectrum element $\hat{\theta}_2 = (\infty s, 0\,s, \infty, 1\,\frac{1}{s}, \emptyset)$.

With the recursively embedded event spectrum any possible pattern of events within a burst can be described. The event pattern consists of a limited set of events that can be repeated by the period of the parent hierarchical event spectrum element.

EXAMPLE 7.2.14. An event pattern with a burst of five events which is repeated after 50 ms and in which the elements within the burst have an intra-arrival rate of 2 ms can be simply described by $\hat{\Theta}_3 = \{(50\,ms, 0\,ms, 5, 0\,\frac{1}{s}, \{(2\,ms, 0\,ms, 1, \infty\,\frac{1}{s}, \emptyset)\})\}$.

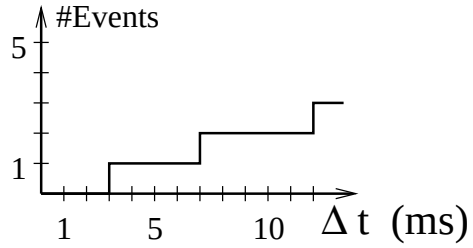

FIGURE 7.2.4. Example simple periodic event sequence

We use the same description model for the limited event pattern as for the repetition of the event pattern. Therefore the event pattern itself can also consist of repeated sub event-patterns that can be described by a sub-sub event spectrum.

EXAMPLE 7.2.15. An event pattern in which the pattern of element $\hat{\Theta}_3$ is repeated 20 times and then a break of about 1000 ms occurs would be described by

$$\hat{\Theta}_4 = \{(2000\,ms, 0\,ms, 100, 0\frac{1}{s}, \hat{\Theta}_3)\}$$

An event spectrum can have several hierarchical levels. On the lowest level only the slope is available to describe the occurring event pattern, but as we have seen, this is sufficient for basic event patterns, even for single events. On the other levels limited periodic repetitions of the event pattern of the sub-hierarchical levels are available. With this concept it is possible to model (and analyze) even complex event patterns efficiently.

CONDITION 7.2.16. For each spectrum element $\hat{\theta}$ either $\hat{\Theta}_{\hat{\theta}} = \emptyset$ or $f_{\hat{\theta}} = 0$

Therefore it is not necessary to distribute the limitation between the slope and the sub-spectrum. This simplifies the analysis without restricting the modeling capabilities.

The period p and the offset a follow the same definition as in the event stream model. So the arrival of the first event occurs after a time units and at $a+p$, $a+2p$, $a+3p$, ..., $a+ip$ the other events occur.

In the following we will give a few examples to show the usage and the possibilities of the new model.

EXAMPLE 7.2.17. A simple periodic event sequence with period $p = 5\,ms$ and offset $a = 2\,ms$ can be modeled by a single event spectrum element:

$$\hat{\Theta}_5 = \{(5\,ms, 2\,ms, 1, \infty\frac{1}{s}, \emptyset\}$$

This example is outlined in figure 7.2.4.

Each periodic event sequence can be directly modeled with the event spectrum model by replacing each previous event element with a spectrum element having the same period and offset and additionally a limitation $L = 1$ and a slope $f = \infty$.

LEMMA 7.2.18. *Let Θ be an event stream with $\Theta = \{\theta_1, ..., \theta_n\}$. The event pattern of Θ is also modeled by a hierarchical event spectrum $\hat{\Theta}$ with $\hat{\Theta} = \{\hat{\theta}_1, ..., \hat{\theta}_n\}$ and with $\hat{\theta}_i = (p_{\theta_i}, a_{\theta_i}, 1, \infty, \emptyset)$*

PROOF. Each of the event stream elements generates exactly one event at each of its periods due to $L = 1$ following the pattern of the corresponding event element. Therefore the complete event spectrum follows the pattern of the event stream. □

EXAMPLE 7.2.19. The same event element as above, but now approximated after 10 events would be modeled in the following way:

$$\hat{\Theta}_5^{10} = \{(\infty s, 0\,ms, 10\,ms, 0\,\frac{ms}{ms}, \{(5\,ms, 2\,ms, 1, \infty\,\frac{1}{s}, \emptyset)\}), (\infty s, 47\,ms, \infty, \frac{1}{5\,ms}, \emptyset)\}$$

Note that $47\,ms = 2\,ms + 5\,ms\,(10-1)$ is the point in time in which the last regular event occurs and therefore the start of the approximation.

There exist two possible concepts for the description of the limitation, an amount L as chosen for the proposed model or the length of the interval in which the events of the sub-spectrum element occur. Having a slope $f = \infty$, like in the basic element, the amount generated cannot be bounded by any interval. Every interval would lead to an infinite amount. Only for an interval of length zero it would be a question of definition whether the interval would lead to an amount of zero or infinity. But no interval would lead to the amount of limitation. The length of the limitation interval can be calculated out of the number of events and the generation pattern using the interval bound function ψ:

$$\Delta t = \psi(L, \hat{\Theta}_{\hat{\theta}}) + L f_{\hat{\theta}}$$

with

$$\psi(L, \emptyset) = 0$$

Note that this calculation requires the condition 7.2.16 (either $f_{\hat{\theta}} = 0$ or $\hat{\Theta}_{\hat{\theta}} = \emptyset$).

CONDITION 7.2.20. *Let $\hat{\theta}_n, \hat{\theta}_{n+i}$ be two event elements with $\hat{\theta}_{n+i}$ be the i-th child of event element $\hat{\theta}_n$. For each possible child i, the child element $\hat{\theta}_{n+i}$ is unequal to the parent element $\hat{\theta}_n$: $\hat{\theta}_n \neq \hat{\theta}_{n+i}$*

Condition 7.2.20 prevents that the recursion can have an infinite depth. The condition also results out of the separation condition, at least for non-trivial cases having not only the same limitations on all levels or not only infinite periods.

7.3. Reduction and normalization of hierarchical event spectra

To allow an easy composition of event spectra and to formulate mathematical operations for the real-time analysis in an easy way a normal form of event spectra is formulated. Also an operator to reduce any spectrum to this normal form is provided. For the normal form we allow only recursively embedded spectra that are either empty or have only one spectrum element.

EXAMPLE 7.3.1. For example an event spectrum $\hat{\Theta} = \{(100\,ms, 0\,ms, 20\,ms, 0\,\frac{ms}{ms}, \hat{\Theta}_a)\}$ with $\hat{\Theta}_a = \{(5\,ms, 0\,ms, 2\,ms, \infty\,\frac{ms}{ms}, \emptyset), (7\,ms, 2\,ms, 3\,ms, 1\,\frac{ms}{ms}, \emptyset)\}$ can be rewritten as $\hat{\Theta} = \{(100\,ms, 0\,ms, 10\,ms, 0\,\frac{ms}{ms}, \hat{\theta}_{a,1}), (100\,ms, 0\,ms, 10\,ms, 0\,\frac{ms}{ms}, \hat{\theta}_{a,2})\}$ with $\hat{\theta}_{a,1} = (5\,ms, 0\,ms, 2\,ms, \infty\,\frac{ms}{ms}, \emptyset)$ and $\hat{\theta}_{a,2} = (7\,ms, 2\,ms, 3\,ms, 1\,\frac{ms}{ms}, \emptyset)$.

LEMMA 7.3.2. *A spectrum* $\hat{\Theta}_A = \{(p_a, a_a, L_a, 0, \hat{\Theta}'_a)\}$ *with a child spectrum element* $\hat{\Theta}'_a = \{(p'_1, a'_1, L'_1, f'_1, \hat{\Theta}_1), ..., (p'_k, a'_k, L'_k, f'_k, \hat{\Theta}_k)\}$ *can be transferred into an equivalent spectrum* $\hat{\Theta}_B$ *with several spectrum elements* $\hat{\Theta}_B = \{\hat{\theta}_{b,1}, \hat{\theta}_{b,2}, ..., \hat{\theta}_{b,n}, \hat{\theta}_{b,x}\}$ *having only child spectra with one element where*

$$\hat{\theta}_{b,i} = (p_a, a_a, \eta(\Delta t_a, \hat{\theta}'_{a,i}), 0, \hat{\theta}'_{a,i})$$

$$\Delta t_a = \lim_{\substack{\varepsilon \to 0 \\ \varepsilon > 0}} (\psi(L_a, \hat{\Theta}'_a) - \varepsilon)$$

$$\hat{\theta}_{b,x} = \left(\infty, \psi(L_a, \hat{\Theta}'_a), L_a - \sum_{\forall \hat{\theta} \in \hat{\Theta}'_a} \eta(\Delta t_a, \hat{\theta}), \infty, \emptyset \right)$$

PROOF. For $\hat{\Theta}_A$ only the second case ($p_{\hat{\theta}} = \infty$, $f_{\hat{\theta}} \neq \infty$) and fourth case ($p_{\hat{\theta}} \neq \infty$, $f_{\hat{\theta}} \neq \infty$) of the spectrum bound function $\eta(\Delta t, \hat{\Theta}_a)$ is relevant (see lemma 7.2.2). Let us consider an event spectrum $\hat{\Theta}_A$ with $p_a = \infty$ first:

$$\begin{aligned}
\eta(\Delta t, \hat{\Theta}_a) &= \min(L_a, (\Delta t - a_a) f_a + \eta(\Delta t - a_a, \hat{\Theta}'_a)) \\
&= \min(L_a, \eta(\Delta t - a_a, \hat{\Theta}'_a)) \\
&= \begin{cases} L_a & \eta(\Delta t - a_a, \hat{\Theta}'_a) \geq L_a \\ \eta(\Delta t - a_a, \hat{\Theta}'_a) & \eta(\Delta t - a_a, \hat{\Theta}'_a) < L_a \end{cases}
\end{aligned}$$

Only for $\eta(\Delta t - a_a, \hat{\Theta}'_a) < L_a$ the sub-spectra of $\hat{\Theta}_a$ are relevant. Then we have

$$\eta(\Delta t - a_a, \hat{\Theta}'_a) = \sum_{\forall \hat{\theta}_{a,i} \in \hat{\Theta}_a} \eta(\Delta t - a_a, \hat{\theta}_{a,i})$$

Let us consider now the corresponding event spectrum $\hat{\Theta}_b$. We have

$$\begin{aligned}
\eta(\Delta t, \hat{\Theta}_b) &= \sum_{i \leq n} \eta(\Delta t, \hat{\theta}_{b,i}) + \eta(\Delta t, \hat{\theta}_{b,x}) \\
&= \sum_{i \leq n} \min(L_{b,i}, \eta(\Delta t - a_a, \hat{\theta}'_{a,i})) + \eta(\Delta t, \hat{\theta}_{b,x})
\end{aligned}$$

The interval Δt_b at which the limitation is reached is the same for each of the spectrum elements. It is $\Delta t_b = \min(\Delta t | \Delta t > \Delta t_a) = \min\left(\Delta t | \Delta t > \lim_{\substack{\varepsilon \to 0 \\ \varepsilon > 0}} (\psi(L_a, \hat{\Theta}'_a) - \varepsilon)\right) = \psi(L_a, \hat{\Theta}'_a)$ and therefore also the same as the interval for which the limitation is reached for $\hat{\Theta}_b$. For $\Delta t < \Delta t_b$ we also know $\eta(\Delta t, \hat{\theta}_{b,x}) = 0$. With this knowledge we get

$$\eta(\Delta t, \hat{\Theta}_b) = \begin{cases} \sum_{i \leq n} L_{b,i} + L_{b,x} & \Delta t \geq \Delta t_b \\ \sum_{i \leq n} \eta(\Delta t - a_a, \hat{\theta}'_{a,i}) + 0 & \Delta t < \Delta t_b \end{cases}$$

For $\Delta t < \Delta t_b$ it is obvious that $\eta(\Delta t, \hat{\Theta}_a) = \eta(\Delta t, \hat{\Theta}_b)$. For $\Delta t \geq \Delta t_b$ we have $\eta(\Delta t, \hat{\Theta}_b) = \sum_{i \leq n} L_{b,i} + L_{b,x} = \sum_{i \leq n} \eta(\Delta t_a, \hat{\theta}'_{a,i}) + (L_a - \sum_{i \leq n} \eta(\Delta t_a, \hat{\theta}'_{a,i}) = L_a = \eta(\Delta t, \hat{\Theta}_a)$.

The proof for the other case ($p_{\hat{\theta}} \neq \infty$, $f_{\hat{\theta}} \neq \infty$) follows in the same way. □

To normalize the event spectrum $\hat{\Theta}_a$ by splitting the child event spectra $\hat{\Theta}'_a$ into it's elements and add one separate event element in the normalized event spectrum $\hat{\Theta}_b$ for each event element of $\hat{\Theta}_a$ we have to distribute the limitation L_a on the elements of the child

event spectra. First we have to find the interval $\Delta t'$ for which the limitation of the parent element L_a is reached by the child event spectrum $\hat{\Theta}'_a$. $\Delta t'$ is given by the interval-bound function $\psi(L_a, \hat{\Theta}'_a)$ for the child event spectrum $\hat{\Theta}'_a$. Then we have to calculate the amount of costs required for each of the child event spectrum elements for $\Delta t'$. This amount of costs is given in general by the event bound function $\eta(\Delta t', \hat{\theta}_i)$ for $\Delta t'$. The problem is that several elements can have a slope of ∞ exactly at the end of this interval. In this situation the sum of the event bound function for all child event elements for $\Delta t'$ may exceed the allowed limitation L_a of the parent element. The total amount of costs generated by these elements is then bounded by the global limitation L_a rather than by their own limitations L'_i. To take this effect into account we exclude the costs occurring exactly at the end of $\Delta t'$ for each hierarchical event element and we handle these costs separately modeling them with the hierarchical event element $\hat{\theta}_{a,x}$. To do so we calculate the limitation not by $\eta(\Delta t', \hat{\theta}'_i)$ but by $\eta(\Delta t' - \varepsilon, \hat{\theta}'_i)$ where ε is an infinite small value excluding only costs occurring exactly at the end of $\Delta t'$.

Another point for the normalization, also resulting out of the separation condition is that the limitation of an embedded event stream element does not exceed the limitation of each of its parent.

CONDITION 7.3.3. *The limitation of a hierarchical event element $\hat{\theta}_n$ $L_{\hat{\theta}_n}$ should not exceed the limitation of any parent event elements $\hat{\theta}_{n-i}$:*

$$\forall \hat{\theta}_{n-i} L_{\hat{\theta}_{n-i}} \leq L_{\hat{\theta}_n}$$

An exceeding limitation would be blocked by the limitation of the parent and therefore would have no effect on the resulting stimuli or cost function. In case the limitation is equal to the limitation of the parent, the period of the child can be set to ∞ as the second period would have also no effect on the resulting stimuli or cost function.

7.4. Capacity Function

The proposed spectrum model can also describe the capacity of resources and allows describing systems with fluctuating capacity over the time. The processor demand analysis and the event stream approach assume that the available amount of resources is the same in each time interval with equal length. Consequently, the workload of the tasks is measured in execution time on the given resource, and the function to model the available amount of resources for each interval length is the intersection. An ideal processor can handle one time unit execution time during one time unit real time. For many resources the capacity is not constant. The reason for a fluctuating capacity can be for example operation-system tasks or variable processor speeds due to energy constraints.

Also the modularization of the analysis requires complex capacities. Consider, for example, a fixed priority scheduling. In a modular approach each priority level gets the remaining capacity of the previous priority level as available capacity. The remaining capacity can be calculated step-wise for each priority level taking only the remaining capacities of the next higher priority level into account. Such an approach is only possible with a model that can describe the remaining capacities exactly.

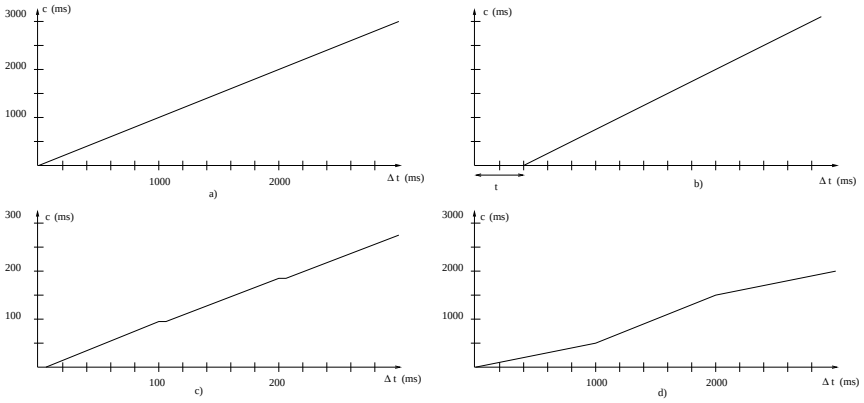

FIGURE 7.4.1. Example service bound functions

DEFINITION 7.4.1. *The spectrum bound function* $\eta(\Delta t, \hat{\Theta}_\rho^-)$ *of a lower spectrum* $\hat{\Theta}_\rho^-$ *of a resource* ρ *gives for each interval* Δt *the minimum amount of processing time that is available for processing tasks in any interval of size* Δt.

An approach covering this concepts are the service curves of the real-time calculus (section 2.3.6).

In the following we will show, with a few examples, how to model fluctuating service functions with the spectrum model.

EXAMPLE 7.4.2. The constant capacity, as shown in 7.4.1 a) can be modeled by a hierarchical event spectrum with only one element:

$$s_{basic} = \{(\infty, 0, \infty, 1, \emptyset)\}$$

EXAMPLE 7.4.3. Blocking the service for a certain time t, as shown in figure 7.4.1 b) is also easy to model using the offset a:

$$s_{block} = \{(\infty, t, \infty, 1, \emptyset)\}$$

EXAMPLE 7.4.4. A constantly growing service curve in which the service is blocked periodically each 100 ms for 5 ms (for example by an task of the operating system), as shown in figure 7.4.1 c), has the following description:

$$s_{pblock} = \{(100\,ms, 5\,ms, 95\,ms, 1\,\frac{ms}{ms}, \emptyset)\}$$

EXAMPLE 7.4.5. The service for a processor that can handle only 1000 ms with full speed and then have to slow down for 1000 ms in which only half the speed is available, as shown in figure 7.4.1 d), is also easy to model:

$$s_{vary} = \{(2000\,ms, 1000\,ms, 500\,ms, \frac{1\,ms}{2\,ms}, \emptyset), (2000\,ms, 0\,ms, 1000\,ms, 1\,\frac{ms}{ms}, \emptyset)\}$$

These are only a few examples how to model complex capacities.

Capacity can be described either by an amount processable within Δt or by a number of events processable within Δt. Operations are required to calculate one form out of the other.

7.5. Modeling common event models with event spectra

To show the universality of the event spectra we will consider in the following section how standard well known event models published in the last years can be described by using event spectra. The idea is to give an impression that event spectra are a fundamental approach to event modeling.

7.5.1. Periodic/sporadic task model with jitter. The most common task model is the periodic / sporadic task model with jitter. The periodic task model was introduced by [88]. In this model the events are occurring strictly periodically with a fixed period p. For this model the maximum event spectrum is given by $\hat{\Theta}^+ = (p, 0, 1, \infty, \emptyset)$. As in the periodic model the events are thought of occurring strictly periodically, the maximum distance between any two consecutive events is one period and therefore the minimum event spectrum in the periodic model is simply $\hat{\Theta}^- = (p, p, 1, \infty, \emptyset)$. Other than in the periodic task model, the maximum distance between the events in the sporadic model is not limited by the period. The sporadic task model requires only a minimum separation distance s between any two consecutive events. Considering the period p and the minimum separation distance between events s $(p = s)$ as equivalent, it is possible to apply the periodic model and its analysis directly on the sporadic model. As only the minimum separation distance is available, the minimum event spectrum $\hat{\Theta}^-$ has to be modeled as $\hat{\Theta}^- = (\infty, \infty, 1, \infty, \emptyset)$. In case that some information about the minimum occurrence of events exists, a better minimum event spectrum $\hat{\Theta}^-$ can be given, other than with the sporadic task model. A widespread extension of these models is the introduction of a jitter j. The arrival of events occurs generally periodically but the events can occur a bit early or later than expected by the period. The interval in which their occurrence can happen is called jitter interval with the length j and is situated around the expected periodic time of occurrence of the event. The jitter is especially valuable for the analysis of distributed systems. The reason is that the finishing time of a task activated by incoming periodic events can vary between the worst-case response time and the best-case response time of the task, therefore the outgoing event stream of the task can be modeled having a jitter of the worst-case response time minus the best-case response time. For a jitter smaller than the period the minimum distance between two event is simply $p - j$ and all following events occur with the distance p in the worst case, therefore $\hat{\Theta}^+ = (\infty, 0, 1, \infty, \emptyset), (p, p - j, 1, \infty, \emptyset)$ and $\hat{\Theta}^- = (p, p + j, 1, \infty, \emptyset)$. For a jitter larger than the period, the events with a regular occurrence somewhere within the jitter interval can all occur at the same time. This number is given by $n = \left\lfloor \frac{j}{p} + 1 \right\rfloor$ therefore the, maximum event spectrum is $\hat{\Theta}^+ = \left\{ \left(\infty, 0, \left\lfloor \frac{j}{p} \right\rfloor, \infty, \emptyset \right), \left(p, p + \left(\frac{j}{p} - \left\lfloor \frac{j}{p} \right\rfloor \right) p, 1, \infty, \emptyset \right) \right\}$. The minimum event spectrum is again $\hat{\Theta}^- = (p, p + j, 1, \infty, \emptyset)$.

7.5.2. Sporadically period task model. Another model proposed in [8] which was specifically designated for single bursts can also be described by event spectra. The event sequence is described by an inner period p_i describing the distance between events within the burst, a number of events n occurring within the burst and an outer period p_o describing the distance between the starts of two consecutive bursts. An event spectrum with one level

of hierarchy is required. The maximum event spectrum is

$$\hat{\Theta}^{+} = \{(p_o, 0, n, 0, \{(p_i, 0, 1, \infty, \emptyset)\})\}$$

the minimum event spectrum is

$$\hat{\Theta}^{-} = \{(p_o, p_o - np_i, n, 0, \{(p_i, p_i, 1, \infty, \emptyset)\})\}$$

.

7.5.3. Periodic task model with initial burst. The assumption of a concurrently occurrence of several events in the periodic task model with jitter is an overestimation for the distributed analysis. For example, consider a chain of two tasks bounded on different resources. The first task τ_1 is activated by a periodic event sequence with jitter and generates an event each time it finishes its execution. With these events the second task on the other resource is activated. Several of the outgoing events of τ_1 cannot occur concurrently as the instances of τ_1 are executed one after another. Therefore the minimum separation time between the activating events of the second task is at least as large as the minimum execution time of the first task.

DEFINITION 7.5.1. (Compare [69]) *Periodic task model with initial burst*

In the periodic task model with initial bursts the occurrence of events is described by an event model $E = (p, j, s)$ with a period p, a jitter j and a minimum separation distance s between any events.

This event model can be covered exactly by the maximum event spectrum $\hat{\Theta}_{max} = \inf(\{(s, 0, 1, \infty, \emptyset)\}, \{(\infty, 0, n, \infty, \emptyset), (p, s, 1, \infty, \emptyset)\})$ with $n = \left\lfloor \frac{j}{p} + 1 \right\rfloor$ and $d = np - j$, and the minimum event spectrum $\hat{\Theta}^{-} = \{(p, p + j, 1, \infty, \emptyset)\}$ which is equivalent to the minimum spectrum of the periodic task model with jitter. For the definition of inf see section 7.6.

7.5.4. Approximation of the real-time calculus. The real-time calculus requires a concrete description of the upper and lower arrival and service curves. One possibility is to approximate each curve by three consecutive segments, one for the initial event, one for the initial burst and one for the overall arrival or service rate, each given by the coordinates x, y of its start point and an event slope s. This approximation can be very easily transferred into a corresponding event spectrum with three elements ($\hat{\Theta} = (\infty, x_1, y_2, s_1, \emptyset), (\infty, x_2, y_3 - y_2, s_2, \emptyset), (\infty, x_3, \infty, s_3, \emptyset)$). The transformation is of course the same for the upper and lower real-time calculus curves into their corresponding event spectrum curves. As this description is very pessimistic a more accurate approximation of the curves is used. The curves consist of an initial non-periodic and a periodic part. Each part is modeled by a set of consecutive line segments. Each line segment w is given by the coordinated x,y of its start point and a slope s. Starting the slope at the starting point leads to the coordinates x',y' of the next following segment w'. The periodic part is described by its starting coordinates x_p, y_p, again by a set of line segments with relative coordinates x,y to the starting point, and by an offset $\Delta x, \Delta y$ between two periods. The transfer to the

event spectrum is easy with:

$$\begin{aligned}\hat{\Theta} = \quad & \{(\infty, x_1, y_1, \infty, \emptyset), (\infty, x_1, s_1(x_2 - x_1), s_1, \emptyset), (\infty, x_2, y_2 - s_1(x_2 - x_1) - y_1, \infty, \emptyset), \ldots, \\ & (\infty, x_n, s_n(x_p - x_n), s_n, \emptyset), (\infty, x_p, y_p - s_n(x_p - x_n) - y_n, \infty, \emptyset), (\Delta x, x_p + x_u, \\ & s_u(x_{u+1} - x_u), s_u, \emptyset), (\Delta x, x_p + x_u, y_u - s_u(x_{u+1} - x_u), s_u, \emptyset), \ldots, (\Delta x, \\ & x_p + x_{u+v}, s_{u+v}(\Delta x - x_v), s_v, \emptyset), (\Delta x, x_p + x_{u+v}, \Delta y - s_{u+v}(\Delta x - x_v) - y_v, \infty, \emptyset)\}\end{aligned}$$

The maximum and the minimum curves are transferred in the same way.

7.6. Event Spectra Algebra

In the following we will introduce some necessary operations on event spectra. By using these operations it is possible to define the real-time analysis in a formal mathematical way on event spectra. Some operations are keeping the properties of upper and/or lower spectra. The step-wise infimum and supremum operator, the add- and scale-operator are examples for this kind of operators. Also the convolution $\otimes$ and deconvolution $\oslash$ belongs to this kind of operators. The other kind of operators is leading only to spectra. The shift operator belongs to this kind. All operators are keeping the properties of a spectrum, which are the monotonic non-decreasing spectrum bound function and the separation condition.

7.6.1. Upper and lower spectra keeping operations. Let us first consider operators keeping the properties of event spectra. This means if the operator is used on upper respectively lower event spectra the resulting spectrum is also an upper respectively a lower spectrum. Then the resulting spectrum keeps the condition of sub- or super-additivity.

7.6.1.1. *Add-operation.* The add operation for two event streams can be simply realized by a merger of the sets of event elements of the two event streams:

DEFINITION 7.6.1. ($+$ operation) If $\hat{\Theta}_C = \hat{\Theta}_A + \hat{\Theta}_B$ then for each interval Δt the equation $\eta(\Delta t, \hat{\Theta}_C) = \eta(\Delta t, \hat{\Theta}_A) + \eta(\Delta t, \hat{\Theta}_B)$ is true.

THEOREM 7.6.2. *($+$ operation) The sum $\hat{\Theta}_C = \hat{\Theta}_A + \hat{\Theta}_B$ can be calculated by the union of the event stream elements of $\hat{\Theta}_A, \hat{\Theta}_B$: $\hat{\Theta}_C = \hat{\Theta}_A \cup \hat{\Theta}_B$*

PROOF.

$$\begin{aligned}\eta(\Delta t, \hat{\Theta}_C) &= \eta(\Delta t, \hat{\Theta}_A) + \eta(\Delta t, \hat{\Theta}_B) \\ &= \sum_{\hat{\theta} \in \hat{\Theta}_A} \eta(\Delta t, \hat{\theta}) + \sum_{\forall \hat{\theta} \in \hat{\Theta}_B} \eta(\Delta t, \hat{\theta}) \\ &= \sum_{\forall \hat{\theta} \in \hat{\Theta}_A \cup \hat{\Theta}_B} \eta(\Delta t, \hat{\theta}) \\ &= \eta(\Delta t, \hat{\Theta}_A \cup \hat{\Theta}_B)\end{aligned}$$

□

LEMMA 7.6.3. *The $+$ operator keeps the properties of upper and lower spectra. So we have $\hat{\Theta}^+ = \hat{\Theta}^+ + \hat{\Theta}^+$ and $\hat{\Theta}^- = \hat{\Theta}^- + \hat{\Theta}^-$. But we also have $\hat{\Theta} = \hat{\Theta} + \hat{\Theta}^+$, $\hat{\Theta} = \hat{\Theta} + \hat{\Theta}^-$ and $\hat{\Theta} = \hat{\Theta}^+ + \hat{\Theta}^-$. Of cause $\hat{\Theta} = \hat{\Theta} + \hat{\Theta}$ is also valid.*

PROOF. Let us consider the case $\hat{\Theta}_C^+ = \hat{\Theta}_A^+ + \hat{\Theta}_B^+$ and two intervals $\Delta t, \Delta t'$. We know due to the sub-additivity that $\eta(\Delta t + \Delta t', \hat{\Theta}_A) \leq \eta(\Delta t, \hat{\Theta}_A) + \eta(\Delta t', \hat{\Theta}_A)$ and $\eta(\Delta t + \Delta t', \hat{\Theta}_B) \leq \eta(\Delta t, \hat{\Theta}_B) + \eta(\Delta t', \hat{\Theta}_B)$ therefore

$$\begin{aligned}\eta(\Delta t + \Delta t', \hat{\Theta}_C^+) &= \eta(\Delta t + \Delta t', \hat{\Theta}_A^+) + \eta(\Delta t + \Delta t', \hat{\Theta}_B^+) \\ &\leq \eta(\Delta t, \hat{\Theta}_A^+) + \eta(\Delta t', \hat{\Theta}_A^+) + \eta(\Delta t, \hat{\Theta}_B^+) + \eta(\Delta t', \hat{\Theta}_B^+) \\ &\leq \eta(\Delta t, \hat{\Theta}_A^+) + \eta(\Delta t, \hat{\Theta}_B^+) + \eta(\Delta t', \hat{\Theta}_A^+) + \eta(\Delta t', \hat{\Theta}_B^+) \\ &\leq \eta(\Delta t, \hat{\Theta}_C^+) + \eta(\Delta t', \hat{\Theta}_C^+)\end{aligned}$$

The other combinations follows in a similar way. □

7.6.1.2. *Infimum (*inf*) and supremum (*sup*).* The other important operator for the min-plus dioid is the step-wise infimum (or minimum) operation, for the max-plus dioid the stepwise supremum (or maximum) operator.

DEFINITION 7.6.4. (inf- infimum (or minimum) operator
If $\hat{\Theta}_C = \inf(\hat{\Theta}_A, \hat{\Theta}_B)$ then for each interval Δt $\eta(\Delta t, \hat{\Theta}_C) = \inf(\eta(\Delta t, \hat{\Theta}_A), \eta(\Delta t, \hat{\Theta}_B))$.

DEFINITION 7.6.5. (sup - supremum (or maximum) operator)
If $\hat{\Theta}_C = \sup(\hat{\Theta}_A, \hat{\Theta}_B)$ then for each interval Δt $\eta(\Delta t, \hat{\Theta}_C) = \sup(\eta(\Delta t, \hat{\Theta}_A), \eta(\Delta t, \hat{\Theta}_B))$.

As

$$\sup(\eta(\Delta t, \hat{\Theta}_A), \eta(\Delta t, \hat{\Theta}_B)) = \sup(\eta(\Delta t, \hat{\Theta}_B), \eta(\Delta t, \hat{\Theta}_A))$$

and

$$\inf(\eta(\Delta t, \hat{\Theta}_A), \eta(\Delta t, \hat{\Theta}_B)) = \inf(\eta(\Delta t, \hat{\Theta}_B), \eta(\Delta t, \hat{\Theta}_A))$$

the operations are commutative and as

$$\sup(\sup(\eta(\Delta t, \hat{\Theta}_A), \eta(\Delta t, \hat{\Theta}_B)), \eta(\Delta t, \hat{\Theta}_C)) = \sup(\eta(\Delta t, \hat{\Theta}_B), \sup(\eta(\Delta t, \hat{\Theta}_A), \eta(\Delta t, \hat{\Theta}_C))$$

as well as

$$\inf(\inf(\eta(\Delta t, \hat{\Theta}_A), \eta(\Delta t, \hat{\Theta}_B)), \eta(\Delta t, \hat{\Theta}_C)) = \inf(\eta(\Delta t, \hat{\Theta}_B), \inf(\eta(\Delta t, \hat{\Theta}_A), \eta(\Delta t, \hat{\Theta}_C))$$

the operations are associative.

For both operators it can be necessary to combine one element of one event spectrum with several elements of the other event spectrum. Therefore these operations can be expensive.

7.6.1.3. *Scaling with a cost value* $(\cdot)$. Another operation on a spectrum is to scale the total spectrum by a cost value. This is for example necessary for the integration of the worst-case execution times into the analysis, so for a transfer of an event-base spectrum to a cost-base spectrum.

DEFINITION 7.6.6. *Let $\hat{\Theta}'$ be the spectrum $\hat{\Theta}$ scaled by the cost value c^+ ($\hat{\Theta}' = c^+\hat{\Theta}$). For each interval Δt the corresponding event bound functions have the relationship*

$$\eta(\Delta t, \hat{\Theta}') = c^+ \eta(\Delta t, \hat{\Theta})$$

LEMMA 7.6.7. *$\eta(\Delta t,\hat{\Theta}') = c^+\eta(\Delta t,\hat{\Theta})$ if the child set of $\hat{\Theta}'$ contains and only contains for each element $\hat{\theta}$ of the child set of $\hat{\Theta}$ an element $\hat{\theta}' \in \hat{\Theta}'$ having the following relations to $\hat{\theta}$: $p_{\hat{\theta}'} = p_{\hat{\theta}}$, $a_{\hat{\theta}'} = a_{\hat{\theta}}$, $n_{\hat{\theta}'} = c^+ n_{\hat{\theta}}$, $\hat{\Theta}_{\hat{\theta}'} = c^+\hat{\Theta}_{\hat{\theta}}$, $f_{\hat{\theta}} = c^+ f_{\hat{\theta}}$*

All parts of the spectra elements related to the number of events are scaled by the variable c^+.

PROOF.

$$c^+\eta(\Delta t,\hat{\Theta}) = \sum_{\substack{\hat{\theta}\in\hat{\Theta}\\ \Delta t\geq a_{\hat{\theta}}}} c^+\eta(\Delta t,\hat{\theta})$$

$$c^+\eta(\Delta t,\hat{\theta}) = \begin{cases} min(c^+L_{\hat{\theta}}, c^+ f_{\hat{\theta}}(\Delta t - a_{\hat{\theta}}) + c^+\eta(\Delta t,\hat{\Theta}_{\hat{\theta}}) & p_{\hat{\theta}} = \infty \\ c^+L_{\hat{\theta}} & |f_{\hat{\theta}}| = \infty \\ \left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rfloor c^+L_{\hat{\theta}} + min(c^+L_{\hat{\theta}}, c^+ f_{\hat{\theta}} mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}) + & \\ \qquad + c^+\eta(mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}),\hat{\Theta}_{\hat{\theta}})) & p_{\hat{\theta}} \neq \infty \end{cases}$$

$$= \begin{cases} min(L_{\hat{\theta}'}, f_{\hat{\theta}'}(\Delta t - a_{\hat{\theta}}) + \eta(\Delta t,\hat{\Theta}_{\hat{\theta}'}) & p_{\hat{\theta}} = \infty \\ L_{\hat{\theta}'} & |f_{\hat{\theta}}| = \infty \\ \left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rfloor L_{\hat{\theta}'} + min(L_{\hat{\theta}}, f_{\hat{\theta}'} mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}) + & \\ \qquad + \eta(mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}),\hat{\Theta}_{\hat{\theta}'})) & p_{\hat{\theta}} \neq \infty \end{cases}$$

$$= \eta(\Delta t,\hat{\theta}')$$

□

The operation is appropriate when having a single worst-case execution time. In realistic systems the execution time of two or more consecutive executions of the same job can be smaller than two or more times the worst-case execution time value. In [95] it was proposed for such systems to model the worst-case execution times as a function of the number of consecutive executions of the jobs having the maximum total costs for two, three, four, ... consecutive executions. One way to combine such an execution-time function with the (hierarchical) event stream model is to do it within the analysis algorithms. The analysis would first calculate the number of events using the (hierarchical) event bound function and after that calculate the amount of execution time required for this number of events.

7.6.2. Convolution and Deconvolution. The real-time calculus relies on the min-plus-dioid $(\mathscr{R}\cup\infty, inf, +)$ and the max-plus-dioid $(\mathscr{R}\cup\infty, sup, +)$. The following operators and proofs for them can be found in [131].

DEFINITION 7.6.8. Min-Plus Convolution $\otimes$ / deconvolution $\oslash$

The min-plus convolution $\hat{\Theta}_C = \hat{\Theta}_A \otimes \hat{\Theta}_B$ and the min-plus convolution $\hat{\Theta}_C = \hat{\Theta}_A \oslash \hat{\Theta}_B$ is given by:

$$\begin{aligned} \eta(\Delta t,\hat{\Theta}_A \otimes \hat{\Theta}_B) &= \inf_{0\leq\Delta t'\leq\Delta t}\{\eta(\Delta t - \Delta t',\hat{\Theta}_A) + \eta(\Delta t',\hat{\Theta}_B)\} \\ \eta(\Delta t,\hat{\Theta}_A \oslash \hat{\Theta}_B) &= \sup_{0\leq\Delta t'<\infty}(\eta(\Delta t + \Delta t',\hat{\Theta}_A) - \eta(\Delta t',\hat{\Theta}_B)\} \end{aligned}$$

DEFINITION 7.6.9. Max-plus Convolution $\bar{\otimes}$ / deconvolution $\bar{\oslash}$

The max-plus convolution $\hat{\Theta}_C = \hat{\Theta}_A \bar{\otimes} \hat{\Theta}_C$ and the max-plus deconvolution $\hat{\Theta}_C = \hat{\Theta}_A \bar{\oslash} \hat{\Theta}_B$ is given by:

$$\eta(\Delta t, \hat{\Theta}_A \bar{\otimes} \hat{\Theta}_B) = \sup_{0 \leq \Delta t' \leq \Delta t} \{\eta(\Delta t - \Delta t', \hat{\Theta}_A) + \eta(\Delta t', \hat{\Theta}_B)\}$$
$$\eta(\Delta t, \hat{\Theta}_A \bar{\oslash} \hat{\Theta}_B) = \sup_{0 \leq \Delta t' \leq \Delta t} \{\eta(\Delta t + \Delta t', \hat{\Theta}_A) - \eta(\Delta t', \hat{\Theta}_B)\}$$

LEMMA 7.6.10. *These convolutions and deconvolutions can be used for obtaining the outgoing arrival and service curves of a greedy processing component (GPC) out of incoming arrival and service curves (see [131] and section 2.3.6 for more details).*

DEFINITION 7.6.11. The outgoing arrival $(\hat{\Theta}^+_{\alpha'}, \hat{\Theta}^-_{\alpha'})$ and the service curves $\hat{\Theta}^+_{\beta'}, \hat{\Theta}^-_{\beta'}$ of a greedy processing component (GPC) can be calculated by:

$$\hat{\Theta}^+_{\alpha'} = \min\{(\hat{\Theta}^+_{\alpha} \otimes \hat{\Theta}^+_{\beta}) \oslash \hat{\Theta}^-_{\beta}, \hat{\Theta}^+_{\beta}\}$$
$$\hat{\Theta}^-_{\alpha'} = \min\{(\hat{\Theta}^-_{\alpha} \oslash \hat{\Theta}^+_{\beta}) \otimes \hat{\Theta}^-_{\beta}, \hat{\Theta}^-_{\beta}\}$$
$$\hat{\Theta}^+_{\beta'} = \max\{(0 \otimes (\hat{\Theta}^+_{\beta} - \hat{\Theta}^-_{\alpha}), 0)\}$$
$$\hat{\Theta}^-_{\beta'} = \max\{(0 \otimes (\hat{\Theta}^-_{\beta} - \hat{\Theta}^+_{\alpha}), 0)\}$$

PROOF. See [131] □

7.6.3. Spectra keeping operation: shift operation $(\leftarrow, \rightarrow)$**.** The shift operation can be realized by adding or subtracting the shift-value from each offset of all top-level elements of the spectrum. When subtracting, the shift value has not necessarily to be equal or smaller than the smallest offset. The spectrum bound function $\eta(\Delta t, \hat{\Theta})$ with $\Delta t \geq 0$ can handle negative offsets despite that negative intervals are not defined.

DEFINITION 7.6.12. *($\rightarrow$ late shift) Let $\hat{\Theta}$ be a hierarchical event spectrum that is shifted right by the value t resulting in the hierarchical event spectrum $\hat{\Theta}' = \hat{\Theta} \rightarrow t$. The event bound functions have the following relationship:*

$$\eta(\Delta t, \hat{\Theta}') = \begin{cases} \eta(\Delta t - t, \hat{\Theta}) & \Delta t \geq t \\ 0 & else \end{cases}$$

LEMMA 7.6.13. $\eta(\Delta t, \hat{\Theta}) \rightarrow t = \eta(\Delta t, \hat{\Theta}')$ *if $\hat{\Theta}'$ contains and only contains for each element $\hat{\theta}$ of $\hat{\Theta}$ an element $\hat{\theta}' \in \hat{\Theta}'$ having the following relations to $\hat{\theta}$: $p_{\hat{\theta}'} = p_{\hat{\theta}}$, $a_{\hat{\theta}'} = a_{\hat{\theta}} + t$, $n_{\hat{\theta}'} = n_{\hat{\theta}}$, $\hat{\Theta}_{\hat{\theta}'} = \hat{\Theta}_{\hat{\theta}}$, $f_{\hat{\theta}'} = f_{\hat{\theta}}$*

The operation $\hat{\Theta}' = \hat{\Theta} \rightarrow t$ can be performed by only adding the value t to the offset $a_{\hat{\theta}}$ for each spectrum element $\hat{\theta} \in \hat{\Theta}$ for its corresponding counter-element $\hat{\theta}' \in \hat{\Theta}'$.

PROOF.

$$\eta(\Delta t - t, \hat{\Theta}) = \sum_{\substack{\hat{\theta} \in \hat{\Theta} \\ \Delta t \geq t}} \eta(\Delta t - t, \hat{\theta})$$

$$
\begin{aligned}
&= \sum_{\substack{\hat{\theta}\in\hat{\Theta} \\ \Delta t \geq a_{\hat{\theta}}+t}} \eta'(\Delta t - t - a_{\hat{\theta}}, \hat{\theta}) \\
&= \sum_{\substack{\hat{\theta}\in\hat{\Theta} \\ \Delta t \geq a_{\hat{\theta}}+t}} \eta'(\Delta t - (a_{\hat{\theta}}+t), \hat{\theta}) \\
&= \sum_{\substack{\hat{\theta}\in\hat{\Theta} \\ \Delta t \geq a_{\hat{\theta}}+t}} \eta'(\Delta t - a_{\hat{\theta}'}, \hat{\theta}) \\
&= \eta(\Delta t, \hat{\Theta}')
\end{aligned}
$$

□

The operation to shift a value left by the value t ($\hat{\Theta} \leftarrow t$) can be defined in a similar way.

DEFINITION 7.6.14. *($\leftarrow$ early shift) Let $\hat{\Theta}$ be a spectrum that is shifted left by the value t resulting in the spectrum $\hat{\Theta}' = \hat{\Theta} \leftarrow t$. The spectrum bound functions have the following relationship:*

$$\eta(\Delta t, \hat{\Theta}') = \eta(\Delta t + t, \hat{\Theta})$$

LEMMA 7.6.15. *$\eta(\Delta t, \hat{\Theta}) \leftarrow t = \eta(\Delta t, \hat{\Theta}')$ if $\hat{\Theta}'$ contains and only contains for each element $\hat{\theta}$ of $\hat{\Theta}$ an element $\hat{\theta}' \in \hat{\Theta}'$ having the following relations to $\hat{\theta}$: $p_{\hat{\theta}'} = p_{\hat{\theta}}$, $a_{\hat{\theta}'} = a_{\hat{\theta}} - t$, $n_{\theta'} = n_{\theta}$, $\hat{\Theta}_{\hat{\theta}'} = \hat{\Theta}_{\hat{\theta}}$, $f_{\hat{\theta}'} = f_{\hat{\theta}}$*

PROOF.

$$
\begin{aligned}
\eta(\Delta t + t, \hat{\Theta}) &= \sum_{\hat{\theta}\in\hat{\Theta}} \eta(\Delta t + t, \hat{\theta}) \\
&= \sum_{\substack{\hat{\theta}\in\hat{\Theta} \\ \Delta t \geq a_{\hat{\theta}}-t}} \eta'(\Delta t + t - a_{\hat{\theta}}, \hat{\theta}) \\
&= \sum_{\substack{\hat{\theta}\in\hat{\Theta} \\ \Delta t \geq a_{\hat{\theta}}-t}} \eta'(\Delta t - (a_{\hat{\theta}}-t), \hat{\theta}) \\
&= \sum_{\substack{\hat{\theta}\in\hat{\Theta} \\ \Delta t \geq a_{\hat{\theta}}-t}} \eta'(\Delta t - a_{\hat{\theta}'}, \hat{\theta}) = \eta(\Delta t, \hat{\Theta}')
\end{aligned}
$$

□

THEOREM 7.6.16. *The operators $(\leftarrow, \rightarrow)$ are associative with the $(+)$ operator and with $\hat{\Theta} = \{\}$ and $t = 0$ as identity elements, so we have $(\hat{\Theta}_A + \hat{\Theta}_B) \rightarrow t = (\hat{\Theta}_A \rightarrow t) + (\hat{\Theta}_B \rightarrow t)$ and $(\hat{\Theta}_A + \hat{\Theta}_B) \leftarrow t = (\hat{\Theta}_A \leftarrow t) + (\hat{\Theta}_B \leftarrow t)$. For $(\hat{\Theta} \rightarrow t) \rightarrow v$ we can write also $\hat{\Theta} \rightarrow (t + v)$.*

PROOF. We will show the proof for $(\hat{\Theta}_A + \hat{\Theta}_B) \rightarrow t = (\hat{\Theta}_A \rightarrow t) + (\hat{\Theta}_B \rightarrow t)$.

$$
\begin{aligned}
(\eta(\Delta t, \hat{\Theta}_A + \hat{\Theta}_B) \leftarrow t &= \eta(\Delta t + t, \hat{\Theta}_A + \hat{\Theta}_B) \\
&= \eta(\Delta t + t, \hat{\Theta}_A) + \eta(\Delta t + t, \hat{\Theta}_B) \\
&= \eta(\Delta t, \hat{\Theta}_A \leftarrow t) + \eta(\Delta t, \hat{\Theta}_B \leftarrow t)
\end{aligned}
$$

The proof for all other conditions follows in a similar way. □

7.6.4. Order of spectra. There are groups of spectra for which the order between the spectra is undecidable. For these groups its impossible to define whether a spectrum is smaller or larger than another spectrum. For other groups of spectra the order is decidable as the relationship between the y-values of two spectra is always the same. For this second group the spectra form a partially ordered set.

DEFINITION 7.6.17. $(\hat{\Theta}_A < \hat{\Theta}_B)$
$\hat{\Theta}_A < \hat{\Theta}_B$ if and only if $\forall \Delta t \geq 0$:

$$\eta(\Delta t, \hat{\Theta}_A) < \eta(\Delta t, \hat{\Theta}_B)$$

DEFINITION 7.6.18. $(\hat{\Theta}_A > \hat{\Theta}_B)$
$\hat{\Theta}_A > \hat{\Theta}_B$ if and only if $\forall \Delta t \geq 0$:

$$\eta(\Delta t, \hat{\Theta}_A) > \eta(\Delta t, \hat{\Theta}_B)$$

DEFINITION 7.6.19. $(\hat{\Theta}_A \leq \hat{\Theta}_B)$
$\hat{\Theta}_A \leq \hat{\Theta}_B$ if and only if $\forall \Delta t \geq 0$:

$$\eta(\Delta t, \hat{\Theta}_A) \leq \eta(\Delta t, \hat{\Theta}_B)$$

DEFINITION 7.6.20. $(\hat{\Theta}_A \geq \hat{\Theta}_B)$
$\hat{\Theta}_A \geq \hat{\Theta}_B$ if and only if $\forall \Delta t \geq 0$:

$$\eta(\Delta t, \hat{\Theta}_A) \geq \eta(\Delta t, \hat{\Theta}_B)$$

DEFINITION 7.6.21. $(\hat{\Theta}_A = \hat{\Theta}_B)$
$\hat{\Theta}_A = \hat{\Theta}_B$ if and only if $\forall \Delta t \geq 0$:

$$\eta(\Delta t, \hat{\Theta}_A) = \eta(\Delta t, \hat{\Theta}_B)$$

THEOREM 7.6.22. *The binary relation over a set of spectra defined in definition 7.6.17 to definition 7.6.21 is reflexive antisymmetric and transitive and therefore a partially ordered set. That means for spectra* $\hat{\Theta}_A, \hat{\Theta}_B, \hat{\Theta}_C$ *we have*

$$\hat{\Theta}_A \leq \hat{\Theta}_A$$
$$\hat{\Theta}_A \leq \hat{\Theta}_B \wedge \hat{\Theta}_B \leq \hat{\Theta}_A \Rightarrow \hat{\Theta}_A = \hat{\Theta}_B$$
$$\hat{\Theta}_A \leq \hat{\Theta}_B \wedge \hat{\Theta}_B \leq \hat{\Theta}_C \Rightarrow \hat{\Theta}_A \leq \hat{\Theta}_C$$

PROOF. The proof for these relationships follows out of the definitions: $\hat{\Theta}_A \leq \hat{\Theta}_A$ as $\eta(\Delta t, \hat{\Theta}_A) = \eta(\Delta t, \hat{\Theta}_A) \Rightarrow \eta(\Delta t, \hat{\Theta}_A) \leq \eta(\Delta t, \hat{\Theta}_A)$. The other relations follows in the same way. □

THEOREM 7.6.23. *The add operation, scale operation,* inf *and* sup *operations and the shift operations keeps the order, therefore for example* $\hat{\Theta}_A \leq \hat{\Theta}_B \Rightarrow (\hat{\Theta}_A \rightarrow x) \leq (\hat{\Theta}_B \rightarrow x)$.

PROOF. As $\eta(\Delta t, \hat{\Theta}_A) \leq \eta(\Delta t, \hat{\Theta}_B) \Rightarrow \eta(\Delta t, \hat{\Theta}_A \rightarrow x) \leq \eta(\hat{\Theta}_B \rightarrow x)$ the proof is obvious for the shift operation. The proof for the other operations follows correspondingly. □

7.6.5. Utilization. An important value for the feasibility analysis is always the utilization of a task set. Let Γ be a task set and let the tasks $\tau \in \Gamma$ be described by a spectrum Θ_τ including the worst-case execution times.

LEMMA 7.6.24. *The utilization U_Γ of a task set in which the event generation patterns are described by spectrum is given by* $((\forall \tau \in \Gamma)\Lambda\,(\forall \hat{\theta} \in \hat{\Theta}_\tau)|(L_{\hat{\theta}} \neq \infty \vee p_{\hat{\theta}} = \infty))$*:*

$$U_\Gamma = \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ p_\tau \neq \infty}} \frac{n_{\hat{\theta}}}{p_{\hat{\theta}}} + \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ L_{\hat{\theta}} = \infty \\ p_{\hat{\theta}} = \infty}} \left(U_{\hat{\Theta}_{\hat{\theta}}} + f_{\hat{\theta}} \right)$$

Note that as in the spectrum-model spectrum-elements with an infinite period do not make any contribution to the utilization. Their contribution gets infinitely small in the long run.

PROOF. The proof is based on the fact that in the long run the contribution of the last period gets infinitely small compared to the rest.

$$\begin{aligned}
U_\Gamma &= \lim_{\Delta t \to \infty} \left(\frac{\eta(\Delta t, \hat{\Theta}_\Gamma)}{\Delta t} \right) \\
&= \lim_{\Delta t \to \infty} \left(\frac{\sum_{\forall \tau \in \Gamma} \sum_{\forall \hat{\theta} \in \hat{\Theta}_\tau} \eta(\Delta t, \hat{\theta})}{\Delta t} \right) \\
&= \sum_{\forall \tau \in \Gamma} \lim_{\Delta t \to \infty} \left(\begin{array}{l} \dfrac{\sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ p_\tau \neq \infty}} \left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rfloor L_{\hat{\theta}} + \min(\ldots)}{\Delta t} + \\ \dfrac{\sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ p_\tau = \infty \\ L_\tau \neq \infty}} min(L_{\hat{\theta}}, (\Delta t - a_{\hat{\theta}}) f_{\hat{\theta}} + \eta(\Delta t - a_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}})}{\Delta t} + \\ \dfrac{\sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ p_\tau = \infty \\ L_\tau = \infty}} (\Delta t - a_{\hat{\theta}}) f_{\hat{\theta}} + \eta(\Delta t - a_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}})}{\Delta t} \end{array} \right) \\
&= \sum_{\forall \tau \in \Gamma} \left(\begin{array}{l} \lim\limits_{\Delta t \to \infty} \dfrac{\sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ p_\tau \neq \infty}} \left(\frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right) L_{\hat{\theta}}}{\Delta t} + \\ \lim\limits_{\Delta t \to \infty} \dfrac{\sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ p_\tau = \infty \\ L_\tau \neq \infty}} L_{\hat{\theta}}}{\Delta t} + \\ \lim\limits_{\Delta t \to \infty} \dfrac{\sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ p_\tau = \infty \\ L_\tau = \infty}} (\Delta t - a_{\hat{\theta}}) f_{\hat{\theta}} + \eta(\Delta t - a_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}})}{\Delta t} \end{array} \right)
\end{aligned}$$

$$= \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ p_\tau \neq \infty}} \frac{n_{\hat{\theta}}}{p_{\hat{\theta}}} + \sum_{\forall \tau \in \Gamma} \sum_{\substack{\forall \hat{\theta} \in \hat{\Theta}_\tau \\ L_{\hat{\theta}} = \infty \\ p_{\hat{\theta}} = \infty}} \left(U_{\hat{\Theta}_{\hat{\theta}}} + f_{\hat{\theta}} \right)$$

□

Even a task set with a low utilization might nevertheless be infeasible. And, of course, task sets having utilization higher than 100% are infeasible too.

7.7. Schedulability analysis

For the schedulability analysis of uni-processor system using the spectrum model, approaches similar to those proposed in chapter 3, 4 and 7 can be used. But with the spectrum model and the operations defined on it we can integrate the approximation and the available capacity into the analysis.

In the following we will show how an exact schedulability test can be realized with the introduced model and operations. We will first discuss the schedulability test for a uni-processor system using EDF (Earliest Deadline First) scheduling. Later we will extend the result to fixed priority scheduled systems.

7.7.1. Schedulability analysis for dynamic priority systems. The general schedulability analysis for EDF is again the processor demand criterion but using the demand bound function for the spectra. Let Γ be a task set completely bounded on the resource ρ that has $\hat{\Theta}_\rho^+$ as upper and $\hat{\Theta}_\rho$ as lower available capacity. Let each task $\tau \in \Gamma$ be a task of the task set with a deadline d_τ activated by an upper event spectrum $\hat{\Theta}_\tau^+$ and a lower event spectrum $\hat{\Theta}_\tau^-$.

A system scheduled with EDF is feasible if for all intervals Δt the demand bound function does not exceed the service function:

$$\delta(\Delta t, \Gamma) \leq \eta(\Delta t, \hat{\Theta}_\rho^-)$$

Both, the demand bound and the service function can be described by and calculated out of hierarchical event spectra.

The overall demand bound function of the task set is the sum of the demand bound functions of the single tasks:

$$\delta(\Delta t, \Gamma) = \sum_{\forall \tau \in \Gamma} \delta(\Delta t, \hat{\Theta}_\tau^+)$$

The demand bound function of a single task can be derived out of the events bound function of this task by shifting this function (or the underlying event stream) by the value of the tasks deadline and scaling it with the tasks execution time:

$$\begin{aligned} \delta(\Delta t, \Gamma) &= \sum_{\forall \tau \in \Gamma} \eta(\Delta t - d_\tau, \hat{\Theta}_\tau) c_\tau^+ \\ \delta(\Delta t, \Gamma) &= \eta\left(\Delta t, \sum_{\forall \tau \in \Gamma} \left(\hat{\Theta}_\tau^+ \rightarrow d_\tau \right) c_\tau^+ \right) \end{aligned}$$

This leads to the test:

$$\eta\left(\Delta t, \sum_{\forall \tau \in \Gamma} \left(\hat{\Theta}_\tau^+ \rightarrow d_\tau\right) c_\tau^+\right) \leq \eta(\Delta t, \rho)$$

or

$$\sum_{\forall \tau \in \Gamma} \left(\hat{\Theta}_\tau^+ \rightarrow d_\tau\right) c_\tau^+ \leq \hat{\Theta}_\rho^-$$

An upper bound for Δt, a maximum test interval, is required to limit the run-time of the test. The concept of the maximum test interval was introduced in section 2.2.3. For the spectrum model one maximum test interval available is the busy period. An upper bound for it is given by:

$$\mathscr{B}(\Gamma) = \min(\Delta t | \chi(\Delta t) \geq \eta\left(\Delta t, \sum_{\forall \tau \in \Gamma} \left(\hat{\Theta}_\tau^+ \rightarrow d_\tau\right) c_\tau^+\right)$$

7.7.2. Response-time calculation for static priority scheduling. In the following we will show how a worst-case response time analysis for scheduling with static priorities can be performed with the new model. This shows the capabilities of the spectrum model and allows the integration of many previous concepts. The request bound function calculates the amount of computation time of a higher priority tasks that can interfere and therefore delays a lower-priority task within an interval Δt. It is closely related to, but a bit different from the product of the spectrum bound function and the worst-case execution time of the task. The difference is the exclusion of events at the end point of the interval. The event spectrum bound function contains all events generated within Δt including the events at the start and the end point of the interval. The request bound function instead only contains the events of the start, not the events of the end point of the interval. For the request bound function the computation time of events becomes relevant after the event has occurred. The computation time of those events occurring exactly at the end of the interval Δt and therefore the events themselves are not relevant for Δt. The request bound function can be calculated using the event bound function in the following way:

$$\rho(\Delta t, \tau) = \lim_{\substack{\Delta \rightarrow \Delta t \\ 0 \leq \Delta < \Delta t}} (\eta(\Delta, \Theta_\tau) c_\tau^+)$$

In the event stream model the difference between the calculation of the event bound function and the request bound function is that for calculating the event bound function the lower ceiling function and for calculating the request bound function the upper ceiling function is used.

For the event spectrum model a similar approach can be used. It is only necessary to handle the cases $\Delta t = 0$ differently than in the calculation of the spectrum bound function:

$$\rho(\Delta t, \Gamma) = \sum_{\forall \tau \in \Gamma} c_\tau^+ \rho(\Delta t, \hat{\Theta}_\tau)$$

with

$$\rho(\Delta t, \hat{\Theta}) = \sum_{\substack{\forall \hat{\theta} \in \hat{\Theta} \\ \Delta t > a_{\hat{\theta}}}} \rho(\Delta t, \hat{\theta})$$

with

$$\rho(\Delta t,\hat{\theta}) = \begin{cases} L_{\hat{\theta}} & p_{\hat{\theta}} = \infty, f_{\hat{\theta}} = \infty \\ \left\lceil \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rceil L_{\hat{\theta}} & p_{\hat{\theta}} \neq \infty, f_{\hat{\theta}} = \infty \\ \min(L_{\hat{\theta}}, f_{\hat{\theta}}(\Delta t - a_{\hat{\theta}}) + \rho(\Delta t - a_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}}) & p_{\hat{\theta}} = \infty, f_{\hat{\theta}} \neq \infty \\ \left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rfloor L_{\hat{\theta}} + & \\ \quad \min(L_{\hat{\theta}}, f_{\hat{\theta}}(\Delta t - a_{\hat{\theta}}) + \rho(mod(\Delta t - a_{\hat{\theta}}, p_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}})) & p_{\hat{\theta}} \neq \infty, f_{\hat{\theta}} \neq \infty \end{cases}$$

With this function it is possible to calculate the worst-case response times for the tasks:

THEOREM 7.7.1. *Let τ be a task scheduling with fixed priority scheduling and $\Gamma_{hp(\tau)}$ be the task set containing all task with a higher priority than τ. The response time $r(\tau_{i,1})$ of the first event of τ_i is given by:*

$$r(\tau_{i,1}) = \min(\Delta t | \eta(\Delta t, \rho) \geq c_\tau^+ + \rho(\Delta t, \Gamma_{hp(\tau)}))$$

The value for Δt can be calculated by a fix-point iteration starting with $\Delta t = c_\tau^+$ and re-inserting the calculated values for Δt into the equation above until the value does not change any more. To calculate the maximum response time it is necessary to do the calculation for all events within the busy period.

The busy period of a task set is the maximum interval in which the resource is completely busy, so in which there exists no idle time for the resource. It can be calculated using the request bound function:

$$\mathscr{B}(\Gamma) = \min(\Delta t | \eta(\Delta t, \rho) \geq \rho(\Delta t, \Gamma))$$

THEOREM 7.7.2. *The worst-case response time of a task τ can be found in the busy period of any task set containing τ and all tasks with a higher priority than τ. It is the maximum response time of all $r(J,\tau)$ where:*

$$r(J,\tau) = \min_{\forall 0 \leq \Delta t < \infty} (\Delta t | \eta(J + \Delta t, \rho) \geq \eta(J,\tau) c_\tau^+ + \rho(J + \Delta t, \Gamma_{hp(\tau)}))$$
$$r(\tau) = max_{\forall 0 \leq J \leq \mathscr{B}(\Gamma)}(r(J,\tau))$$

J is lower or equal to the busy period ($J \leq \mathscr{B}(\Gamma)$). The only point remaining is to check for every task if this minimum response time is lower than the deadline of the task ($\forall \tau \in \Gamma | d_\tau \geq r_\tau$). If this is the case the task set activated by event spectra and scheduling with fixed priority scheduling is feasible.

7.7.3. End-to-End Response time analysis. Having a distributed system in which tasks activate other tasks, the end-to-end worst-case response time for a whole task chain is often required. This is the time an event requires at most from occurring in the activating event stream until its result is available at the end of the task chain. One possibility is to calculate the worst-case response-times for each task separately and then add the resulting response times. In situations, in which the response time of an event is large due to delays by previous events of the same event stream this calculation can lead to pessimistic results. The worst-case response times of the following task is calculated taking the upper outgoing event spectrum of the previous task (with the highest density) into account whereas the

worst-case response time of this previous task would lead to an event spectrum with a lower density. Or, in other words, the worst-case response-time of the following task allows such a late arrival of its event (due to the delays by previous events) that the worst-case response-time of the previous task does not play a role. The task chain $\tau_6 \rightarrow \tau_2$ given in figure 9.0.1 is an example for such an situation. It is discussed in more detail in the case study section.

THEOREM 7.7.3. *Let task τ_1 to τ_n be a chain of consecutive tasks activated by the event spectrum $\hat{\Theta}$. Let $r_{x,j}$ denote the worst-case response-time of the j-th event on task x and let $r_{1,j\rightarrow x,j}$ denote the total response time of the j-th event from τ_1 to τ_x. The total worst-case response time of the i-th instance of an event spectrum $\hat{\Theta}$ for a task chain τ_1 to τ_n $(n > 1)$ is given by the maximum of*

$$r_{1,1\rightarrow n,i} = \max_{\forall 1 \leq j \leq i} \{(r_{n,i} - r_{n,j-1} + r_{n-1,j}), (r_{n,j} + r_{n-1,i-j+1})\}$$

PROOF. Each end-to-end response time of the task chain can be split into a response time R_n on τ_n and a response time $R_{1\rightarrow n-1}$ on the remaining task chain. In a first case where the response time $R_n \leq r_n$, the total response time is bounded by $r_n + r_{1\rightarrow n-1,1}$. In case $R_n > r_n$ a delay has to occur somewhere within R_n. The reason for this delay can only be the late arrival of an event at task τ_n due to the response-times on this task. □

7.7.4. Response time analysis for TDMA and RR. In the following we will show the transfer of the calculation to further scheduling approaches. As examples we will consider time division multiple access (TDMA) and the round robin (RR) scheduling. TDMA divides the available processing time into a fixed set of single slots repeated with a period p. Each task has access to one or more of these slots. Deriving the upper and lower service bound out of a TDMA schedule is easy. Having a period of p and a slot-length s leads to an upper service bound of $\hat{\Theta}^+ = (p, 0, s, 1, \emptyset)$ and a lower service bound of $\hat{\Theta}^- = (p, p-s, s, 1, \emptyset)$. In the best case the event occurs just at the start of its slot, therefore having the processor for the next s time units and then repeatedly again after p time units. In the worst-case the event occurs exactly when the slot is finished and it processing is therefore delayed by $p - s$ time units. The round-robin scheduling checks the tasks in a fixed order whether they have available jobs for execution. In such a case these jobs are executed for at most the slot length for the task. Same as with TDMA a job can be distributed on several slots and several jobs can be executed within one slot. Other than with TDMA, having nothing to execute for a task does not lead to an idle time for the processor but the slots for the following tasks are brought forward. Therefore the available capacity for one task depends on the incoming event spectra for the other tasks in the RR-cycle. Each time unit not used by one of the tasks reduces all response times in which the time unit occurs of all other tasks by one time unit. The worst-case response time for RR is similar to the worst-case response time analysis for static priorities, only the calculation of the higher priority processing time differs. This can be calculated by "simulating" the worst-case RR approach on the interval-base event spectrum model.

7.7.5. Hierarchical Scheduling. Different scheduling approaches can be combined to a hierarchical scheduling. The child scheduling approaches can be considered as tasks

of their parent scheduling approach. The top-most parent scheduling gets the completely available capacity of its resource and distribute this capacity on its child scheduling approaches. The distribution follows the same schema as if these child scheduling approaches where simple tasks. The child scheduling approaches get theses distributed capacity as their incoming service event spectrum and can then be considered as being bounded on a resource of its own with a bit more complicated capacity. They can also have other embedded child scheduling approaches and so on. In the example 9.0.1 the fixed-priority scheduling between the tasks τ_1 and task τ_2 is embedded into a TDMA scheduling. Both scheduling approaches can be handled separately. The fixed-priority scheduling gets the outgoing capacity bound function for its TDMA cycle as incoming capacity bound function.

7.8. Limitations of the hierarchical event stream model

The problem with the hierarchical event stream model defined so far and also with the previous concrete description for event spectra, especially with the exact description of the real-time calculus, is that the resulting description size of operations on them depends on the concrete values of their parameters. Especially for task sets with different periods for the tasks, the resulting description can depend on the hyper-period of these periods. As the run-time of the operations and the analysis depends directly on the size of the description it can become quite large.

To solve this problem and to limit the number of elements of the resulting hierarchical event streams the approximation introduced in the previous chapters can be applied also to the hierarchical event stream model. In chapter 8 we will present an efficient way to approximate the hierarchical event spectra which allows an efficient analysis.

CHAPTER 8

Approximation of hierarchical event spectra

The hierarchical event spectrum model allows to generalize the approximation as introduced in chapter 3 to 5. With this model it is possible to integrate the approximation into the curves themselves and therefore to transfer each event spectrum into an approximated event spectrum without leaving the model. For the analysis function as the event spectrum bound function the approximated curves expands only to a limited set of test intervals even without considering upper bounds as, for example, the busy period. The approximated curves are also represented by an event spectrum without further knowledge of the approximation and the chosen degree of exactness. Therefore it is possible to use the same operations of the event spectral algebra (see section 7.6) for the approximated curves as for the non-approximated curves, propagate the approximated curves without knowledge of the approximation error and even combine approximated curves with different degrees of exactness and with exact curves.

This approach is especially important for the modularized analysis of distributed systems. It allows calculating the event spectra of a module using only the calculated event spectra of the previous modules. This is an abstraction from how these previous spectra have been calculated and whether an approximation was used or not.

In the following we will show a general way to transfer event spectra into approximated event spectra bounded with a given degree of exactness. The maximum approximation error for such spectra is guaranteed and for the analysis of these spectra only a strictly limited number of values have to be considered. Like in the superposition approximation for event streams the approximation is done for each event spectrum element separately.

In the following we will introduce two attempts to integrate the approximation in the event spectrum model. In the first one each element is approximated independently of its position in the hierarchy and especially independent of its parent and child elements. Therefore each period of the element leads to a restart of the approximation for the child elements and to a new set of test intervals. In case that the number of test intervals for an element within one period of the parent element is smaller than the number of exactly considered test intervals by the approximation, only the original element without approximation would be used. This is a straight forward way for doing the approximation, but the number of exactly considered values required for such approximated curves depends on the level of hierarchy. In the second attempt the number of test intervals is bounded for each element globally. Therefore in one period of the parent element the exact element of the child is used, in another period of the parent element an approximated version of the child element is taken. With this approximation the number of test intervals is bounded

globally depending only on the number of elements of the original event spectrum and the choosen degree of exactness.

8.1. First approach: Separate approximation for each element

First we will consider to approximate each event spectrum element on each hierarchical level separately. The approximation of each element is done independently of the child and parent element. The general idea of the approximation is to analyze for each element the smallest *n* test intervals exactly and approximate the larger test intervals using the specific utilization of the hierarchical event spectrum element.

8.1.1. Method. Let us first consider the approximation for a simple event spectrum element.

LEMMA 8.1.1. *Let $\hat{\Theta}$ be an event spectrum with $\hat{\Theta} = \{\hat{\theta}_1, ..., \hat{\theta}_n\}$ and let $k = 2\left\lceil \frac{1}{\varepsilon} \right\rceil$. $\hat{\Theta}^k$ is an approximation of $\hat{\Theta}$ with $\varepsilon = \frac{1}{k}$ and $\eta(\Delta t, \hat{\Theta}) \leq \eta(\Delta t, \hat{\Theta}^k) \leq (1+\varepsilon)\eta(\Delta t, \hat{\Theta})$ if for each element $\hat{\theta}_i = \{p_{\hat{\theta}_i}, a_{\hat{\theta}_i}, L_{\hat{\theta}_i}, f_{\hat{\theta}_i}, \hat{\Theta}_{\hat{\theta}_i}\}$ of $\hat{\Theta}$ a set of elements $\{\hat{\theta}^k_{i,1}, \hat{\theta}^k_{i,2}, \hat{\theta}^k_{i,3}\}$ exists in $\hat{\Theta}^k_i$ with*

$$\hat{\theta}^k_{i,1} = (\infty, 0, kL_{\hat{\theta}_i}, 0, \{(p_{\hat{\theta}}, a_{\hat{\theta}}, L_{\hat{\theta}}, f_{\hat{\theta}}, \hat{\Theta}^k_{\hat{\theta}})\})$$
$$\hat{\theta}^k_{i,2} = (\infty, kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}, L_{\hat{\theta}}, \infty, \emptyset)$$
$$\hat{\theta}^k_{i,3} = (\infty, kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}, \infty, \frac{L_{\hat{\theta}_i}}{p_{\hat{\theta}_i}}, \emptyset)$$

PROOF. For the proof we have to distinguish two cases. We have to distinguish between intervals $\Delta t < kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}$ and between intervals $\Delta t \geq kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}$. For both cases we have to prove that the event bound function of the approximated event spectrum is always between the event bound function and $(1+\varepsilon)$-times the event bound function of the exact event spectrum.

Case 1: For the first case only the first of the three event spectrum elements $\hat{\Theta}^k_{i,1}$, $\hat{\Theta}^k_{i,2}$,$\hat{\Theta}^k_{i,3}$ is relevant as the offset of the other elements is larger than Δt. The offset of both other elements $kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}$ is larger than the considered intervals and therefore the resulting event bound function of these elements is always zero. We get for the approximated event spectrum element for the first case ($\Delta t < kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}$) the value:

$$\begin{aligned}\eta(\Delta t, \hat{\theta}^k_i) &= \eta(\Delta t, \hat{\theta}^k_{i,1}) + \eta(\Delta t, \hat{\theta}^k_{i,2}) + \eta(\Delta t, \hat{\theta}^k_{i,3}) \\ &= min(L_{\hat{\theta}^k_{i,1}}, f_{\hat{\theta}^k_{i,1}}(\Delta t - a_{\hat{\theta}^k_{i,1}}) + \eta(\Delta t - a_{\hat{\theta}^k_{i,1}}, \hat{\Theta}_{\hat{\theta}^k_{i,1}})) + 0 + 0 \\ &= min(kL_{\hat{\theta}_i}, 0 + \eta(\Delta t, \hat{\theta}_i))\end{aligned}$$

As $\eta(\Delta t, \hat{\theta})$ is a monotonic non-decreasing function and

$$\begin{aligned}\eta(kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}, \hat{\theta}_i) = \left\lfloor \frac{kp_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \right\rfloor L_{\hat{\theta}_i} + min(L_{\hat{\theta}_i}, f_{\hat{\theta}_i} mod(kp_{\hat{\theta}_i}, p_{\hat{\theta}_i}) + \\ \eta(mod(kp_{\hat{\theta},}, p_{\hat{\theta}_i}), \hat{\theta}_i)) \\ = kL_{\hat{\theta}_i} + min(L_{\hat{\theta}_i}, 0 + 0) \\ = kL_{\hat{\theta}_i}\end{aligned}$$

the value of $\eta(\Delta t', \hat{\theta}_i)$ is for $\Delta t < kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}$ always smaller than $kL_{\hat{\theta}_i}$, therefore

$$\eta(\Delta t, \hat{\theta}_i^k) \quad = \quad \eta(\Delta t, \hat{\theta}_i)$$

Of course this trivial result fulfills the approximation condition ($\eta(\Delta t, \hat{\theta}_i) \leq \eta(\Delta t, \hat{\theta}_i^k) \leq (1+\varepsilon)\eta(\Delta t, \hat{\theta}_i)$)

2. Case: ($\Delta t \geq kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}$)

The event spectrum function $\eta(\Delta t, \hat{\theta}^k)$ for the approximated event element includes contributions of all three elements of the approximated element:

$$\begin{aligned}\eta(\Delta t, \hat{\theta}_i^k) =& \eta(\Delta t, \hat{\theta}_{i,1}^k) + \eta(\Delta t, \hat{\theta}_{i,2}^k) + \eta(\Delta t, \hat{\theta}_{i,3}^k) \\ =& min(kL_{\hat{\theta}_i}, \eta(\Delta t - kp_{\hat{\theta}_i} - a_{\hat{\theta}_i}, \hat{\theta}_i)) + \\ & min(L_{\hat{\theta}_i}, \infty) + min(\infty, \frac{L_{\hat{\theta}_i}}{p_{\hat{\theta}_i}}(\Delta t - kp_{\hat{\theta}_i} - a_{\hat{\theta}_i}))\end{aligned}$$

Due to the monotonic non-decreasing behavior of η and due to the separation condition we have $\eta(\Delta t, \hat{\theta}_i) \geq \eta(kp_{\hat{\theta}_i} + a_{\hat{\theta}_i}, \hat{\theta}_i) \geq kL_{\hat{\theta}_i}$. Therefore we get:

$$\begin{aligned}\eta(\Delta t, \hat{\theta}_i^k) &= kL_{\hat{\theta}_i} + L_{\hat{\theta}_i} + \frac{L_{\hat{\theta}_i}}{p_{\hat{\theta}_i}}(\Delta t - kp_{\hat{\theta}_i} - a_{\hat{\theta}_i}) \\ &= kL_{\hat{\theta}_i} + L_{\hat{\theta}_i} + \Delta t\frac{L_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} - kL_{\hat{\theta}_i} - a_{\hat{\theta}_i}\frac{L_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \\ &= L_{\hat{\theta}_i} + (\Delta t - a_{\hat{\theta}_i})\frac{L_{\hat{\theta}_i}}{p_{\hat{\theta}_i}}\end{aligned}$$

It is now necessary to prove the approximation condition:

$$\eta(\Delta t, \hat{\theta}_i) \leq \eta(\Delta t, \hat{\theta}_i^k) \leq (1+\varepsilon)\eta(\Delta t, \hat{\theta}_i)$$

We have

$$\begin{aligned}\eta(\Delta t, \hat{\theta}_i) &= \left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rfloor L_{\hat{\theta}_i} + min(L_{\hat{\theta}_i}, f_{\hat{\theta}_i} mod(\Delta t - a_{\hat{\theta}_i}, p_{\hat{\theta}_i}) + \eta(mod(\Delta t - a_{\hat{\theta}_i}.p_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}_i}) \\ &\leq \left\lfloor \frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right\rfloor L_{\hat{\theta}_i} + L_{\hat{\theta}_i} \\ &\leq \left(\frac{\Delta t - a_{\hat{\theta}}}{p_{\hat{\theta}}} \right) L_{\hat{\theta}_i} + L_{\hat{\theta}_i} = \eta(\Delta t, \hat{\theta}_i^k)\end{aligned}$$

The proof for unequation $(1+\varepsilon)\eta(\Delta t, \hat{\theta}_i) \geq \eta(\Delta t, \hat{\theta}^k)$ of lemma 8.1.1 is similar:

$$\begin{aligned}(1+\varepsilon)\eta(\Delta t, \hat{\theta}_i) &= (1+\varepsilon)\left(\left\lfloor \frac{\Delta t - a_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \right\rfloor L_{\hat{\theta}_i} + min(...) \right) \\ &\geq \left\lfloor \frac{\Delta t - a_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \right\rfloor L_{\hat{\theta}_i} + \varepsilon \left\lfloor \frac{\Delta t - a_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \right\rfloor L_{\hat{\theta}_i} \geq \left\lfloor \frac{\Delta t - a_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \right\rfloor L_{\hat{\theta}_i} + \frac{2}{k} \left\lfloor \frac{kp_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \right\rfloor L_{\hat{\theta}_i} \\ &\geq \left\lfloor \frac{\Delta t - a_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \right\rfloor L_{\hat{\theta}_i} + 2L_{\hat{\theta}_i} \geq \left(\frac{\Delta t - a_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \right) L_{\hat{\theta}_i} + L_{\hat{\theta}_i} \\ &\geq L_{\hat{\theta}_i} + (\Delta t - a_{\hat{\theta}_i})\frac{L_{\hat{\theta}_i}}{p_{\hat{\theta}_i}} \geq \eta(\Delta t, \hat{\theta}_i^k)\end{aligned}$$

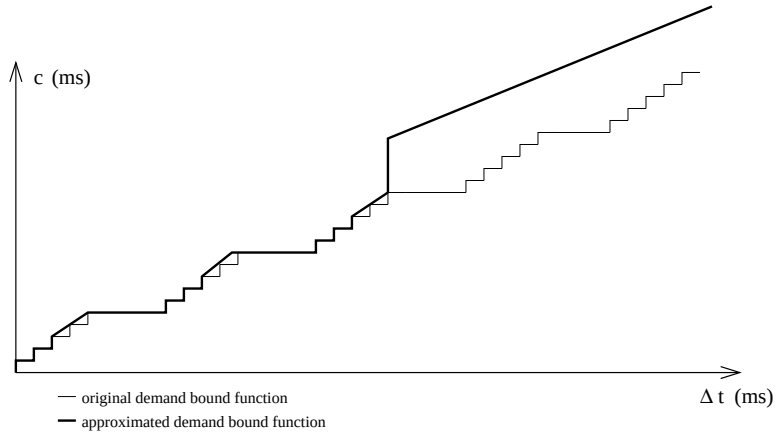

FIGURE 8.1.1. Approximated hierarchical spectrum bound function

We have proven that the proposed approximation for one event spectrum element is sufficient. □

EXAMPLE 8.1.2. Consider for example fig. 8.1.1 that shows the spectrum bound function of the hierachical event spectrum

$$\Theta_7 = \{(20\,ms, 0\,ms, 10\,ms, 0\,\frac{s}{s}, (2\,ms, 0\,ms, 2\,ms, \infty\,\frac{s}{s}, \emptyset))\}$$

.

The hierarchical spectrum consists of bursts with five events. Each event needs two ms computation time and the distance between the events within the bursts is also two ms. The bursts occur periodically with a period of 20 ms. The distance between the end of one burst and the begin of the next following burst is 10 ms. The approximated hierarchical event spectrum with an approximation after three events has the following description:

$$\begin{aligned}\hat{\Theta}^3_{7,b} =& \{(\infty s, 0\,ms, 6\,ms, 0\,\frac{s}{s}, (2\,ms, 0\,ms, 2\,ms, \infty\,\frac{s}{s}, \emptyset)), (\infty s, 4\,ms, 4\,ms, 1\,\frac{s}{s}, \emptyset)\}\\ \hat{\Theta}^3_7 =& \{(\infty s, 0\,ms, 30\,ms, 0\,\frac{s}{s}, (20\,ms, 0\,ms, 10\,ms, 0\,\frac{s}{s}, \hat{\Theta}^3_{7,b})),\\ & (\infty s, 48\,ms, 10\,ms, \infty\,\frac{s}{s}, \emptyset), (\infty s, 48\,ms, \infty s, \frac{1\,ms}{2\,ms}, \emptyset)\}\end{aligned}$$

The original hierarchical spectrum element is split up into three separate parts. The first element $\hat{\theta}^3_{7,1} = (\infty s, 0\,ms, 30\,ms, 0\,\frac{s}{s}, (20\,ms, 0\,ms, 10\,ms, \hat{\Theta}^3_{7,b}))$ models the non- approximated part of the original element. It is equal to the non-approximated hierarchical spectrum element except that the approximation boarder kL_θ limits its length and that the approximative description $\hat{\Theta}^3_{7,b}$ of the child spectrum element is used. The second spectrum element $\hat{\theta}^3_{7,2} = (\infty s, 48\,ms, 10\,ms, \infty\,\frac{s}{s}, \emptyset)$ adds one time the limitation $L_{\hat{\theta}_7}$. The approximation can be done for each spectrum element of the hierarchical spectrum separately. The exact distribution of the costs within the intervals depends on the child spectra and/or the slope and is therefore unknown when considering only the hierarchical spectrum element itself. We have to assume for the approximation as worst-case situation that the costs occur completely at the start of the interval. To make sure that the approximated spectrum bound

function does always meet or exceed the original spectrum bound function the second spectrum element is required.

The third spectrum element $\hat{\theta}^3_{7,3} = (\infty s, 48\,ms, \infty s, \frac{1\,ms}{2\,ms}, \emptyset)$ models the approximated part of the spectrum. Like in chapter 3 it is done by the specific utilization as slope.

The maximum amount of costs for which the approximated and the original event bound function for one event stream element of the event stream model can differ is one event (see chapter 3). For the hierarchical event spectrum model the maximum difference between the approximated and the original spectrum bound function for an hierarchical spectrum element $\hat{\theta}$ with no child spectrum element is one time the limitation $L_{\hat{\theta}}$. For a hierarchical event spectrum element with a child spectrum element this amount can be larger as the exact distribution of the limitation within the child spectrum element is unknown. This missing knowledge of the exact distribution is compensated by the second spectrum element $\hat{\theta}^k_{i,2}$ by adding one time $L_{\hat{\theta}}$. So the difference can be bounded by $2L_{\hat{\theta}}$. The relative error resulting of this difference is bounded by $\varepsilon = \frac{2L_{\hat{\theta}}}{kL_{\hat{\theta}}} = \frac{2}{k}$ as for k test intervals the spectrum bound function includes at least $kL_{\hat{\theta}}$ events. This error remains the same for complete hierarchical event spectrum even with several levels of hierarchy. To achieve the same relative error, as with the event stream model it is required to consider $2k$ test intervals exactly.

8.1.2. Complexity. Substituting each original spectrum element with its approximative counterpart consisting of the three approximated spectrum elements limits the required number of test intervals. For hierarchical spectra this number depends not only on the number of elements and the chosen approximation error, but also on the maximum level of hierarchy. In the cases that the hierarchical spectrum element $\hat{\theta}$ has a recursively embedded event spectrum $\hat{\Theta}_{\hat{\theta}}$ this spectrum element is evaluated for each test interval of $\hat{\theta}$ from the beginning. Therefore for each test interval of $\hat{\theta}$ which is not approximated, the child event spectrum $\hat{\Theta}_{\hat{\theta}}$ contributes a whole set of test intervals. For this approximation of $\hat{\theta}$ only the period $p_{\hat{\theta}}$ and the limitation $L_{\hat{\theta}}$ is relevant, not the concrete shape of the embedded sub event spectrum $\hat{\Theta}_{\hat{\theta}}$.

As the child spectrum can again contain hierarchical spectrum elements with own child spectra, the necessary number of test intervals can increase substantially. Having a system with a sum of n spectrum elements and a maximum recursion level of d, the maximum number of test intervals in its approximated description with error ε is bounded by $k = (n\lceil\frac{2}{\varepsilon}\rceil)^d$.

EXAMPLE 8.1.3. The problem is illustrated with the following example:

$$\hat{\theta}_8 = \{1000\,ms, 0\,ms, 150\,ms, 0\frac{s}{s}, \{\hat{\theta}_9\}\}$$
$$\hat{\theta}_9 = \{10\,ms, 0\,ms, 5\,ms, \infty\frac{s}{s}, \emptyset\}$$

We have a child spectrum element $\hat{\theta}_9$ with 30 test intervals in each period of its parent spectrum element $\hat{\theta}_8$. We allow the approximation after 100 test-intervals. The approximation of $\hat{\theta}_8^{100}$ using lemma 8.1.1 reads as follows:

$$\hat{\theta}_8^{100} = \{(\infty s, 0\,s, 15\,s, 0, \hat{\theta}_8), (\infty, 100\,s, 150\,ms, \infty \frac{s}{s}, \emptyset), (\infty s, 100\,s, \frac{150\,ms}{1000\,ms}, \emptyset)\}$$

The total number of test-intervals needed for evaluation the spectrum bound function of this event spectrum is $100 + 100 \cdot 30 = 3100$.

8.2. Second approach: Global approximation for each element

To limit the number of test intervals further we propose another approach. The approximation of a spectrum element starts after the necessary number of test intervals is reached globally for this spectrum element. The start of the approximation is independent of in which period of the parent spectrum element the necessary number of test intervals is reached. In this approach we generally split the parent spectrum element into the part in which the child spectrum is approximated and the part in which it is modeled exactly. So in case that the event element $\hat{\theta}$ is a child element of another (parent) event element $\hat{\theta}'$ we have to distinguish for $\hat{\theta}'$ between those periods in which $\hat{\theta}$ is evaluated exactly and those in which $\hat{\theta}$ is approximated. To do this it is necessary to split $\hat{\theta}'$ at the last exactly considered interval of $\hat{\theta}$.

8.2.1. Simple event spectrum element. Let us consider first a simple hierarchical event element:

$$\hat{\theta} = \{(p, a, L, f, \emptyset)\}$$

LEMMA 8.2.1. *$\hat{\theta}^k$ is the approximative counter-part for $\hat{\theta} = \{(p, a, L, f, \emptyset)\}$ starting with the approximation after k exactly considered test intervals. $\hat{\theta}^k$ is modeled by:*

$$\begin{aligned}\hat{\theta}^k = &\{(\infty, 0, L_A, 0, \hat{\theta}),\\ &(\infty, a_A, L, f, \emptyset), (\infty, a_B, \infty, \frac{L}{p}, \emptyset)\}\end{aligned}$$

with

$$\begin{aligned}L_A &= kL\\ a_A &= a + kp\\ a_B &= a_A + \frac{L}{f}\end{aligned}$$

For the special case with $f = \infty$ we have $a_A = a_B$.

PROOF. We have to prove that $\eta(\Delta t, \hat{\Theta}) \leq \eta(\Delta t, \hat{\Theta}^k)$ and that

$$\frac{\eta(\Delta t, \hat{\Theta}^k) - \eta(\Delta t, \hat{\Theta})}{\eta(\Delta t, \hat{\Theta})} \leq \frac{1}{k}\eta(\Delta t, \hat{\Theta}^k) \leq \frac{k+1}{k}\eta(\Delta t, \hat{\Theta})$$

Let us first consider the intervals up to the approximation, so $\Delta t < a + kp$

$$\eta(\Delta t, \hat{\theta}^k) = \min(\eta(\Delta t, \hat{\theta}), kL) + 0 + 0$$

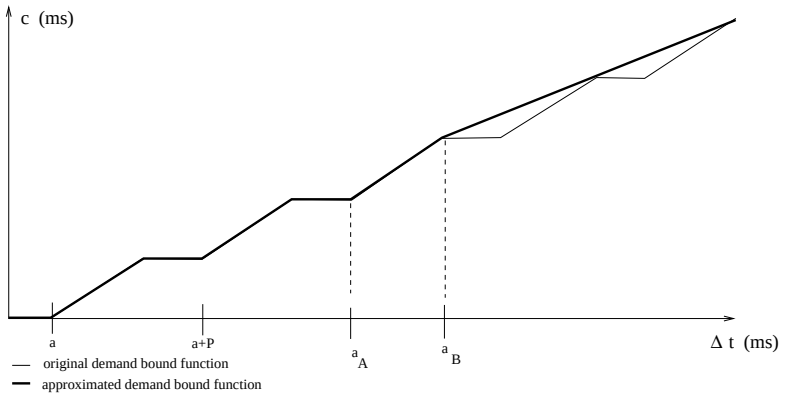

FIGURE 8.2.1. Case one simple event spectrum element

$$= \eta(\Delta t, \hat{\theta})$$

as $\eta(a+kp, \hat{\theta}) \leq kL$ as the function is monotonically rising.

For $a+kp \leq \Delta t < a+kp+\frac{L}{f}$ the second element of the approximated event spectrum becomes relevant. Let $\Delta t' = \Delta t - (a+kp)$. Then $\Delta t' < \frac{L}{f}$ and with the separation condition $\Delta t' < p$:

$$\begin{aligned}
\eta(\Delta t, \hat{\theta}) &= \left\lfloor \frac{(\Delta t' + a + kp) - a}{p} \right\rfloor L + \min(\Delta t' f, L) \\
&= \left\lfloor \frac{\Delta t'}{p} \right\rfloor L + kL + \min(\Delta t' f, L) \\
&= kL + \min((\Delta t - a - kp) f, L) + 0 \\
&= \eta(\Delta t, \hat{\theta}^k)
\end{aligned}$$

For $\Delta t \leq a_A$ both functions are equal so both conditions are fulfilled. For $a_A \leq \Delta t \leq a_B$ we have $\eta(\Delta t, \hat{\Theta}) = kL + min(L, (\Delta t - kp) f)$ and $\eta(\Delta t, \hat{\Theta}^k) = L_A + min(L, (\Delta t - a_A) f) = \eta(\Delta t, \hat{\Theta})$.

The remaining proof is visualized in figure 8.2.1. For $\Delta t > a_B$ we have $\eta(\Delta t, \hat{\Theta}) \leq \eta(\Delta t, \hat{\Theta}^k)$ and $\eta(\Delta t, \hat{\Theta}^k) - \eta(\Delta t, \hat{\Theta})$ is bounded by L. As $\eta(a_B, \hat{\Theta}) = (k+1)L$ we have $\frac{\eta(\Delta t, \hat{\Theta}^k) - \eta(\Delta t, \hat{\Theta})}{\eta(\Delta t, \hat{\Theta})} \leq \frac{L}{(k+1)L} \leq \frac{1}{k}$. □

EXAMPLE 8.2.2. Let us, for example consider the event spectrum

$$\hat{\Theta} = \{(10\,ms, 0\,ms, 3\,ms, \frac{1\,ms}{2\,ms}, \emptyset)\}$$

.

The approximation $\hat{\Theta}^5$ for $\hat{\Theta}$ after $k = 5$ exactly considered test intervals is given by

$$\begin{aligned}
\hat{\Theta}^5 = \{&(\infty s, 0\,ms, 15\,ms, 0\,\frac{s}{s}, \{(10\,ms, 0\,ms, 3\,ms, \frac{1\,ms}{2\,ms}, \emptyset)\}), \\
&(\infty s, 50\,ms, 3\,ms, \frac{1\,ms}{2\,ms}, \emptyset), (\infty s, 56\,ms, \infty s, \frac{3\,ms}{10\,ms}, \emptyset)\}
\end{aligned}$$

where

$$L_A = 5 \cdot 3\,ms = 15\,ms$$

$$a_A = 0\,ms + 5 \cdot 10\,ms = 50\,ms$$
$$a_B = 50\,ms + \frac{3\,ms}{\frac{1\,ms}{2\,ms}} = 56\,ms$$

We can simplify to:

$$\hat{\Theta}^5 = \{(\infty s, 0\,ms, 18\,ms, 0\frac{s}{s}, \{(10\,ms, 0\,ms, 3\,ms, \frac{1\,ms}{2\,ms}, \emptyset)\}), (\infty s, 56\,ms, \infty s, \frac{3\,ms}{10\,ms}, \emptyset)\}$$

EXAMPLE 8.2.3. Consider another example event spectrum

$$\hat{\Theta} = \{(10\,ms, 2\,ms, 3\,ms, \infty\frac{s}{s}, \emptyset)\}$$

with $f = \infty$. The approximation $\hat{\Theta}^{10}$ after $k = 10$ exactly considered test intervals is given by:

$$\begin{aligned}\hat{\Theta}^{10} = &\{(\infty s, 0\,ms, 30\,ms, 0\frac{s}{s}, \{(10\,ms, 2\,ms, 3\,ms, \infty\frac{s}{s}, \emptyset)\},\\ &(\infty s, 103\,ms, 3\,ms, \infty\frac{s}{s}, \emptyset), (\infty s, 103\,ms, \infty s, \frac{3\,ms}{10\,ms}, \emptyset)\}\end{aligned}$$

or shorter:

$$\hat{\Theta}^{10} = \{(\infty s, 0\,ms, 33\,ms, 0\frac{s}{s}, \{(10\,ms, 2\,ms, 3\,ms, \infty\frac{s}{s}, \emptyset)\}, (\infty s, 103\,ms, \infty s, \frac{3\,ms}{10\,ms}, \emptyset)\}$$

8.2.2. Approximation of one-level child element. Let us consider an event spectrum with one child element:

$$\hat{\theta} = (p, a, L, 0, \hat{\theta}')$$
$$\hat{\theta}' = (p', a', L', f', \emptyset)$$

LEMMA 8.2.4. *$\hat{\theta}^k$ is the approximative counter-part for $\hat{\theta} = \{(p, a, L, 0, \hat{\theta}')\}$ with $\hat{\theta}' = \{(p', a', L', f', \emptyset)\}$ starting with the approximation after k exactly considered test intervals. $\hat{\theta}^k$ is modeled by:*

$$\begin{aligned}\hat{\Theta}^k = &\{(\infty, 0, L_A, 0, \hat{\theta}^\circ),\\ &(\infty, a_A, kL - L_A, 0, \{(p, a', L', 0, \{\\ &(\infty, 0, L', f', \emptyset), (p, \frac{L'}{f'}, L - L', \frac{L'}{p'}, \emptyset)\})\},\\ &(\infty, a_B, \Delta y, \infty, \emptyset), (\infty, a_B, \infty, \frac{L}{p}, \emptyset)\}\end{aligned}$$

or shorter by

$$\begin{aligned}\hat{\Theta}^k = &\{(\infty, 0, L_A, 0, \hat{\theta}^\circ),\\ &(\infty, a_A, kL - L_A, 0, \{(p, a', L', f', \emptyset),\\ &(p, a' + \frac{L'}{f'}, L - L', \frac{L'}{p'}, \emptyset)\}),\\ &(\infty, a_B, \Delta y, \infty, \emptyset), (\infty, a_B, \infty, \frac{L}{p}, \emptyset)\}\end{aligned}$$

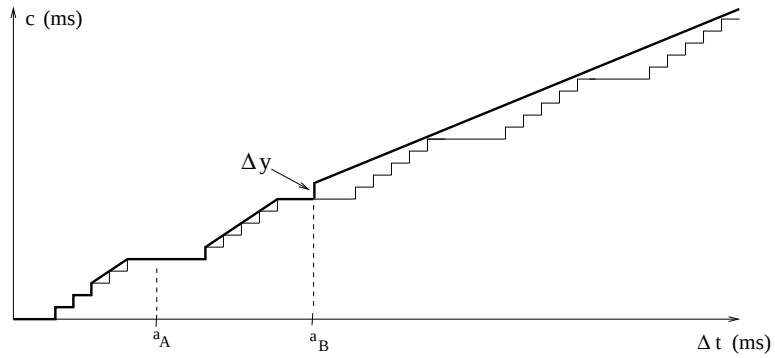

FIGURE 8.2.2. One-level event spectrum element

with

$$\hat{\theta}^{\circ} = \begin{cases} \hat{\theta} & L \leq kL' \\ \{(p,0,L,0,\hat{\theta}'^{k})\} & L > kL' \end{cases}$$

$$\hat{\theta}'^{k} = \{(\infty,0,kL',0,\hat{\theta}),(\infty,kp',L',f',\emptyset),(\infty,kp'+\frac{L'}{f'},\infty,\frac{L'}{p'},\emptyset)\}$$

$$L_A = \begin{cases} \left\lceil \frac{kL'}{L} \right\rceil L & L \leq kL' \\ L & L > kL' \end{cases}$$

$$a_A = \begin{cases} \left\lceil \frac{kL'}{L} \right\rceil p + a & L \leq kL' \\ p + a & L > kL' \end{cases}$$

$$a_B = kp + a + a'$$

$$\Delta y = L - \frac{p'L^2}{pL'} + \frac{p'L}{p} - \frac{a'L}{p}$$

PROOF. The proof for this lemma is visualized in figure 8.2.2. Δy is a cost-offset required to ensure that the spectrum bound function of the approximated spectrum is always equal or greater than the spectrum bound function of the exact spectrum. The limitation of the parent spectrum element can be reached by the child spectrum element somewhere within the period of the parent spectrum element. This can happen early or late within the period of the parent spectrum element.

The first spectrum element of the approximated spectrum $\hat{\Theta}^k$ models the part in which the child spectrum element $\hat{\theta}'$ of the exact spectrum is considered exactly. In case that the first possible approximation interval for $\hat{\theta}'$ occurs within the first period of $\hat{\theta}$, we have to start the approximation within this first period of $\hat{\theta}$. Otherwise it would not be possible to find a reasonable bound for the number of considered test intervals for $\hat{\theta}'$. So $\hat{\theta}^{\circ}$ depends on whether $L \leq kL'$ or $L > kL'$.

The approximation of $\hat{\theta}'$ can be done by an element $\hat{\theta}'^{k}$ with a slope $f_{\hat{\theta}'^{k}} = \frac{L'}{p'}$. This element has a value close to the value of the original spectrum bound function of $\hat{\theta}'$ each time when the limitation of $\hat{\theta}'$ is reached. The element restarts for every period of $\hat{\theta}$. We separate between the approximation of every first event of $\hat{\theta}'$ and the remaining events of

$\hat{\theta}'$. The first events are approximated using the period of $\hat{\theta}$, so only the element covering the remaining events has to restart for every period of $\hat{\theta}$.

When starting finally the approximation of $\hat{\theta}$ a cost-offset Δy is required to ensure that the approximated function $\eta(\Delta t, \hat{\theta}^k)$ is always equal or higher than the exact spectrum bound function $\eta(\Delta t, \hat{\theta})$. This cost-offset is necessary as a new period of the parent spectrum element splits the approximation of the child spectrum element. The calculation of Δy is visualized in figure 8.2.3 and can be done as follows:

$$\begin{aligned} L - \Delta y &= \Delta x \frac{L}{p} \\ \Delta y &= L \frac{p - \Delta x}{p} \\ \Delta y &= L \left(1 - \frac{\Delta x}{p}\right) \end{aligned}$$

Δx gives the interval between the start of the child spectrum element $\hat{\theta}'$ and the point of time in which the limitation of $\hat{\theta}$ is reached. The reaching of the limitation is calculated using the approximative description of the child elements of $\hat{\theta}$ with the separate consideration of every first event of $\hat{\theta}$. For a simple child element $\hat{\theta} = \{(p, a, L, 0, \hat{\theta}')\}$ with $\hat{\theta}' = \{(p', a', L', \infty, \emptyset)\}$ this value y is given by

$$\begin{aligned} (\Delta x - a') \cdot (\frac{L'}{p'}) &= L - L' \\ \Delta x &= \frac{L - L'}{\frac{L'}{p'}} + a' = p' \frac{L}{L'} - p' + a' \end{aligned}$$

Therefore we get for Δy:

$$\Delta y = L - \frac{p' L^2}{p L'} + \frac{p' L}{p} - \frac{a' L}{p}$$

When we additionally approximate every first event of the child spectrum we get the following description:

$$\begin{aligned} \hat{\Theta}^k = &\{(\infty, 0, L_A, 0, \hat{\theta}^\circ), \\ &(\infty, a_A, kL - L_A, 0, \{(\infty, a', L', f', \emptyset), \\ &(\infty, a', \infty, \frac{L'}{p}, \emptyset), (p, a' + \frac{L'}{f'}, L - L', \frac{L'}{p'}, \emptyset)\}), \\ &(\infty, a_B, \Delta y, \infty, \emptyset), (\infty, a_B, \infty, \frac{L}{p}, \emptyset)\} \end{aligned}$$

□

EXAMPLE 8.2.5. Let us consider the example hierarchical spectrum:

$$\begin{aligned} \hat{\Theta} &= \{(80\,ms, 2\,ms, 16\,ms, 0\,\frac{s}{s}, \hat{\Theta}')\} \\ \hat{\Theta}' &= \{(10\,ms, 2\,ms, 3\,ms, \infty\,\frac{s}{s}, \emptyset)\} \end{aligned}$$

For the approximation $\hat{\Theta}^{10}$ with at least $k = 10$ test intervals considered exactly we get the values:

$$L_A = \left\lceil \frac{kL'}{L} \right\rceil L = \left\lceil \frac{10 \cdot 3ms}{16ms} \right\rceil 16ms = 32ms$$
$$a_A = \left\lceil \frac{kL'}{L} \right\rceil p + a = \left\lceil \frac{10 \cdot 3ms}{16ms} \right\rceil 80ms + 2ms = 162ms$$
$$a_B = kp + a + a' = 10 \cdot 80ms + 2ms + 2ms = 804$$
$$\Delta x = p' \frac{L}{L'} - p' + a' = 10ms \frac{16ms}{3ms} - 10ms + 2ms = 45.3333ms$$
$$\Delta y = L\left(1 - \frac{\Delta x}{p}\right) = 16ms\left(1 - \frac{45.333ms}{80ms}\right) = 6.9333ms$$

The spectrum can be written as:

$$\begin{aligned}\hat{\Theta}^{10} =& \{(\infty s, 0ms, 32ms, 0\frac{s}{s}, \{(80ms, 2ms, 16ms, 0\frac{s}{s}, \{(10ms, 2ms, 3ms, \infty\frac{s}{s}, \emptyset)\})\}),\\ & (\infty s, 162ms, 128ms, 0\frac{s}{s}, \{(\infty s, 2ms, 3ms, \infty\frac{s}{s}, \emptyset), \{\infty s, 2ms, \infty s, \frac{3ms}{80ms}, \emptyset),\\ & (80ms, 2ms, 13ms, \frac{3ms}{10ms}, \emptyset)\},\\ & (\infty s, 804ms, 6.9333ms, \infty\frac{s}{s}, \emptyset), (\infty s, 804ms, \infty s, \frac{16ms}{80ms}, \emptyset)\}\end{aligned}$$

8.2.3. Approximation of two-level child element. Let us consider the following hierarchical spectrum element with two levels of child spectral elements:

$$\hat{\theta} = \{(p, a, L, 0, \hat{\theta}')\}$$
$$\hat{\theta}' = \{(p', a', L', 0, \hat{\theta}'')\}$$
$$\hat{\theta}'' = \{(p'', a'', L'', f'', \emptyset)\}$$

We consider the approximation $\hat{\theta}^k$.

LEMMA 8.2.6. *$\hat{\theta}^k$ is given by*

$$\begin{aligned}\hat{\theta}^k =& \{(\infty, 0, L_A, 0, \hat{\theta}^\circ{}_1), (\infty, a_A, L_B, 0, \hat{\theta}^\circ{}_2),\\ & (\infty, a_B, L_C, 0, \{(p, a, L, 0, \{(\infty, a', \Delta y', \infty, \emptyset), (p, a', \infty, \frac{L'}{p'}, \emptyset)\})\}),\\ & (\infty, a_C, \Delta y, \infty, 0), (\infty, a_C, \infty, \frac{L}{p}, \emptyset)\}\end{aligned}$$

$\hat{\theta}^\circ{}_1$ depends on whether $L' \leq kL''$ or $L' > kL''$. We have

$$\hat{\theta}^\circ{}_1 = \begin{cases} \hat{\theta} & L' \leq kL'' \\ \{(\infty, 0, L', 0, \hat{\theta}''^k)\} & L' > kL'' \end{cases}$$

with $\hat{\theta}''^k$ as the stand-alone approximated event spectrum of $\hat{\theta}''$

$$\hat{\theta}''^k = \{(\infty, 0, kL'', 0, \hat{\theta}), (\infty, kp'', L'', f'', \emptyset), (\infty, kp'' + \frac{L''}{f''}, \infty, \frac{L''}{p''}, \emptyset)\}$$

$\hat{\theta}^{\circ}{}_1$ depends on whether $L \leq kL'$ or $L > kL'$. We have

$$\hat{\theta}^{\circ}{}_2 = \begin{cases} \emptyset & nL \leq kL'' < kL' \leq (n+1)L \\ \{(p', 0, L', 0, \hat{\theta}''_a)\} & L \leq kL' \\ \{(\infty, 0, kL' - L_A, 0, \{(p', 0, L', 0, \hat{\theta}''_a)\}), & \\ (\infty, kp' - a_A, L - kL', 0, \hat{\theta}'_a)\} & kL' < L \end{cases}$$

where $\hat{\theta}''_a$ is the approximated part of $\hat{\theta}''$:

$$\hat{\theta}''_a = \{(\infty, a'', L'', f'', 0), (\infty, a'' + \frac{L''}{f''}, \infty, \frac{L''}{p''}, \emptyset)\}$$

and $\hat{\theta}'_a$ is the approximated part of $\hat{\theta}'$:

$$\hat{\theta}'{}_a = \{(\infty, a', \Delta y', \infty, 0), (\infty, a' + \frac{L'}{f'}, \infty, \frac{L'}{p'}, \emptyset)\}$$

The calculation of L_A, a_A and L_B is easy and straight forward:

$$L_A = \begin{cases} L' & kL'' < L' \\ \left\lceil \frac{kL''}{L'} \right\rceil L' & L' \leq kL'' < L \\ \left\lceil \frac{kL''}{L} \right\rceil L & L < kL'' \end{cases}$$

$$L_B = \begin{cases} \left\lceil \frac{kL'}{L} \right\rceil L - L_A & L \leq kL' \\ L - L_A & L > kL' \end{cases}$$

$$L_C = kL - (L_A + L_B)$$

$$a_A = \begin{cases} \left\lceil \frac{kL''}{L} \right\rceil p + a' + a & L \leq kL'' \\ \left\lceil \frac{kL''}{L'} \right\rceil p' + a' + a & L > kL'' \end{cases}$$

$$a_B = \begin{cases} \left\lceil \frac{kL'}{L} \right\rceil p & L \leq kL' \\ p & L > kL' \end{cases}$$

$$a_C = kp + a$$

The calculation of $\Delta y'$ is the same as the calculation for Δy in the previous section. We have

$$\Delta x' = p'' \frac{L'}{L''} - p'' + a''$$

$$\Delta y' = L' \left(\frac{p' - \Delta x'}{p'} \right)$$

The calculation of Δy and Δx is similar but using the approximation of $\hat{\theta}''$. We have

$$(\Delta x - a) \cdot (\frac{L'}{p'}) = L - \Delta y'$$

$$\Delta x = \frac{L - \Delta y'}{\left(\frac{L'}{p'}\right)} + a'$$

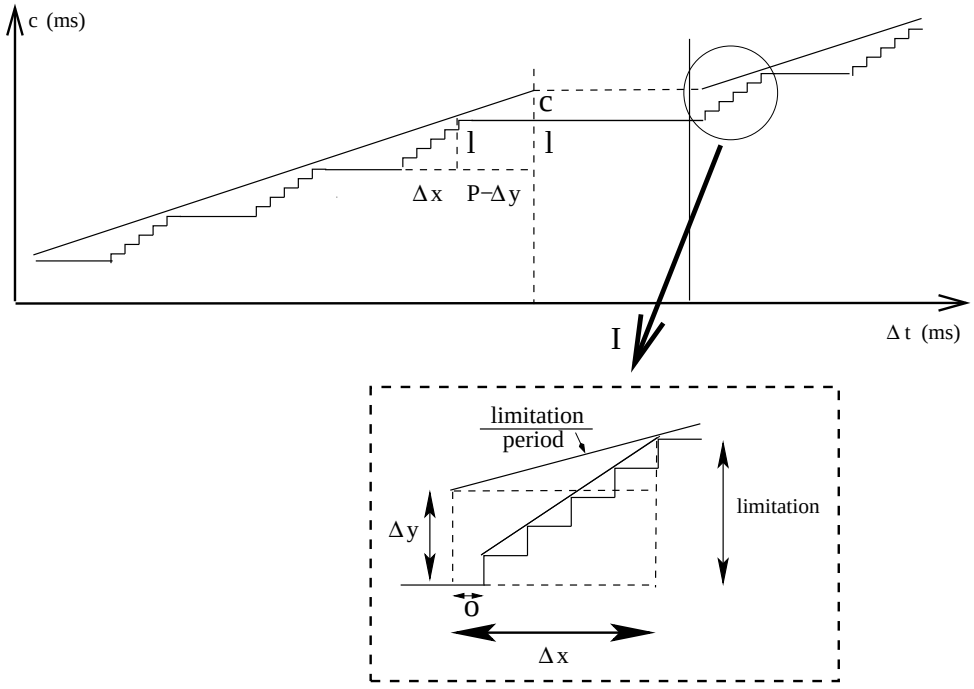

FIGURE 8.2.3. Approximation for hierarchical event spectra

$$\Delta x = \frac{Lp'}{L'} - \frac{\Delta y' p'}{L'} + a'$$
$$\Delta y = L\left(\frac{p-\Delta x}{p}\right)$$

PROOF. The first spectrum element $(\infty, 0, L_A, 0, \hat{\theta}^\circ{}_1)$ models the part of the result in which the spectrum element $\hat{\theta}''$ is completely not approximated (Case $L' < kL''$) or is approximated in a way as if it exists alone. For the second spectrum element $(\infty, a_A, L_B, 0, \hat{\theta}^\circ{}_2)$ only the approximated part of $\hat{\theta}''$ is used. Here $\hat{\theta}'$ is either completely not approximated or handled as if only $\hat{\theta}'$ and $\hat{\theta}''$ exists. The proofs for the first and second element are the same as in the previous section.

The following elements are the same as in the one-level case except for the calculation Δy and $\Delta y'$. Therefore the proof for the one-level case can be used also here. The calculation of Δy is visualized in figure 8.2.3. When setting $\Delta y'' = L''$ the calculation of $\Delta y'$ and $\Delta x'$ on the one side and Δy and Δx on the other side are the same. The proof for the correctness of these values follows directly out of their calculation. Therefore the proposed description for $\hat{\Theta}^k$ can be generalized to handle hierarchical event spectra with n-level child event spectra.

□

In the following we will give a small example by extending the example 8.2.5.

EXAMPLE 8.2.7. Let us consider the example hierarchical event spectrum $\hat{\Theta}$:

$$\hat{\Theta} = \{(1000\,ms, 10\,ms, 100\,ms, 0\frac{s}{s}, \hat{\Theta}')\}$$

$$\hat{\Theta}' = \{(80\,ms, 2\,ms, 16\,ms, 0\frac{s}{s}, \hat{\Theta}'')\}$$
$$\hat{\Theta}'' = \{(10\,ms, 2\,ms, 3\,ms, \infty\frac{s}{s}, \emptyset)\}$$

For an approximation $\hat{\Theta}^{10}$ in which $k = 10$ test intervals are considered exactly we get the values:

$$\Delta x' = \frac{16\,ms - 3\,ms}{\left(\frac{3\,ms}{10\,ms}\right)} + 2\,ms = 45.3333\,ms$$
$$\Delta y' = 16\,ms \cdot \left(\frac{80\,ms - 45.3333\,ms}{80\,ms}\right) = 6.9333\,ms$$
$$\Delta x = \frac{100\,ms - 6.9333\,ms}{\left(\frac{16\,ms}{80\,ms}\right)} + 2\,ms = 467.333\,ms$$
$$\Delta y = 100\,ms \cdot \left(\frac{1000\,ms - 67.3335\,ms}{1000\,ms}\right) = 53.2667\,ms$$

$\hat{\Theta}^{10}$ can be written as:

$$\begin{aligned}\hat{\Theta}^{10} =& \{(\infty s, 0\,ms, 32\,ms, 0\frac{s}{s}, \hat{\Theta}), (\infty s, 162\,ms, 168\,ms, 0\frac{s}{s}, \{(80\,ms, 0\,ms, 16\,ms, 0\frac{s}{s},\\ & \{(\infty s, 2\,ms, 3\,ms, \infty\frac{s}{s}, \emptyset), (\infty s, 2\,ms, \infty\,ms, \frac{3\,ms}{80\,ms}, \emptyset)\})\},\\ & (\infty s, 2010\,ms, 800\,ms, 0\frac{s}{s}, \{(\infty s, 2\,ms, 6.9333\,ms, \infty\frac{s}{s}, \emptyset),\\ & (\infty s, 2\,ms, \infty s, \frac{6.9333\,ms}{1000\,ms}, \emptyset), (10\,s, 2, 93.0667\,ms, \frac{16\,ms}{80\,ms}, \emptyset)\})\\ & , (\infty s, 10010\,ms, 53.2667\,ms, \infty\frac{s}{s}, \emptyset), (\infty s, 10010\,ms, \infty s, \frac{100\,ms}{1000\,ms}, \emptyset)\}\end{aligned}$$

8.2.4. Approximation of n-level spectra. Let us now consider the approximation of normalized event spectra with more than two levels of hierarchy.

DEFINITION 8.2.8. An element $\hat{\theta}_i$ is a subsequent child of an element $\hat{\theta}_1$, if it is related to $\hat{\theta}_1$ in such a way that a chain exists with $\hat{\theta}_1 \rightarrow ... \rightarrow \hat{\theta}_i$ where $\hat{\theta}_j \rightarrow \hat{\theta}_{j+1}$ denotes that $\hat{\theta}_j$ is parent of $\hat{\theta}_{j+1}$.

For each level we have to distinguish between the non-approximated and the approximated parts. An event spectrum element which covers for one subsequent child element both, intervals for which this subsequent child element is approximated and intervals for which it is not approximated, has to be split into two event spectrum elements, so that one of these two covers all non-approximated and one of these two all approximated intervals of the sub-sequence child element. Not only the element alone, but all of its parent elements may need to be split, too. So first it is necessary to calculate these separation points, which are the interval bounds at which the parent element is split to distinguish between the non-approximated and the approximated part of one of its child elements. In the following we give an approach leading to a limited number of new elements but still keeping the approximation.

Let us consider a parent element $\hat{\theta}$ with a subsequent child element $\hat{\theta}'$. In general in this approach, $\hat{\theta}$ is split to cover the approximation of $\hat{\theta}'$ at the first of its completed periods which is larger than the first possible approximation interval of $\hat{\theta}'$.

In those cases in which the approximation of the child element starts within the completion of the first period of the parent element we cannot postpone it until the first period of the parent. It would not be possible to limit the number of test intervals for the child hierarchical event element.

EXAMPLE 8.2.9. Consider the following example:

$$\hat{\theta}_{10} = \{10\,s, 0\,ms, 4000\,ms, 0\frac{s}{s}, \{\hat{\theta}_{11}\}\}$$
$$\hat{\theta}_{11} = \{10\,ms, 0\,s, 5\,ms, \infty\frac{s}{s}, \emptyset\}$$

Again the approximation may start after 100 test-intervals. The approximated event element can be written as follows:

$$\begin{aligned}
\hat{\theta}_{10}^{100} =& \{(\infty s, 0\,ms, 4000\,ms, 0\,ms, \{\hat{\theta}_{11}^{100}\}),\\
& (\infty s, 10\,s, 396\,s, 0, \{(\infty s, 0\,ms, 5\,ms, \infty\frac{s}{s}, \emptyset), (\infty s, 0\,ms, \infty s, \frac{5\,ms}{10000\,ms}, \emptyset),\\
& (10\,s, 0\,s, 3995\,ms, \frac{5\,ms}{10\,ms}, \emptyset)\}),\\
& (\infty s, 1000\,s, 804\,ms, \infty\frac{s}{s}, \emptyset), (\infty s, 1000\,s, \infty s, \frac{4\,s}{10\,s}, \emptyset)\}\\
\hat{\theta}_{11}^{100} =& \{(\infty s, 0\,ms, 500\,ms, 0\frac{s}{s}, \{(10\,ms, 0\,ms, 5\,ms, \infty\frac{s}{s}, \emptyset)\}), (\infty s, 1000\,ms, 5\,ms, \infty\frac{s}{s}, \emptyset),\\
& (\infty s, 1000\,ms, \infty s, \frac{5\,ms}{10\,ms}, \emptyset)\}
\end{aligned}$$

Postponing the approximation of the child up to the end of the first period of the parent would cost 3000 additional test intervals.

Still we only need to split the parent element into two parts, one for the first period and one for all following periods. For the first period before the splitting point we use the child in its original approximated description having an exact and an approximated part. For the following periods only the approximated part of the child $\hat{\theta}_{11,3}^{100}$, where it is described by the slope, is required.

In cases in which the parent element has again a parent element, the periods of the upper-most parent are determining the splitting points for all of its descendants in our approach, so for its child element, the child element of its child element and so on. Only in those cases in which the first possible approximation point of a hierarchical event element $\hat{\theta}$ falls within the first period of this top-level parent element $\hat{\theta}'''$ the splitting point is determined by the upper-most parent element $\hat{\theta}''$ for which the first approximation point $t_{\hat{\theta}}$ of the child element $\hat{\theta}$ falls within the second or later period of the parent element $\hat{\theta}''$. If there isn't any such parent element, the splitting point is the first period of the direct parent of the child element. The child element is substituted by its combined exact and approximative description. This propagation of the splitting point to the top-most parent does not change the upper bound of the required maximum number of test intervals.

8.2.5. Approximation of elements with several heterogeneous child elements. To handle the approximation of an element with several heterogeneous child elements we

can normalize these elements. The normalization is introduced in section 7.3. For the normalization it is necessary to distribute the limitation of the parent element. The result is an event spectrum in which only the upper-most parent spectrum can contain several event elements. Every of these parent elements can than be approximated seperately using the introduced methods.

8.2.6. Required number of test intervals. The purpose of the approximation is to limit the number of test intervals resulting from a hierarchical event spectrum.

There exists a simple bound on the required number of test intervals. For those cases in which the approximation is postponed, so it does not start within the first period, the number of test intervals for one period of the parent event element has to be less than the approximation bound k. Otherwise the approximation would be allowed somewhere within the first period. Therefore the maximum number of test intervals we have to additionally consider due to the postponing is bounded also by k, leading to a total bound of $2k$.

LEMMA 8.2.10. *Delaying the start of the approximation $t_{\hat{\theta}}$ for a hierarchical event element $\hat{\theta}$ up to the first period of the upper-most parent $\hat{\theta}'$ of $\hat{\theta}$ for which $t_{\hat{\theta}} > a_{\hat{\theta}} + p_{\hat{\theta}}$ leads to a required number of test intervals for $\hat{\theta}$ which is bounded by $2k$ where k is the chosen number of test intervals for the approximation.*

PROOF. In all cases in which the top-most parent $\hat{\theta}'$ determines the splitting point for a child $\hat{\theta}_{child}$ less than k test intervals of the child occurs within one period of $\hat{\theta}_{topparent}$. Otherwise the first possible approximation interval would be within the first period of $\hat{\theta}_{topparent}$. Therefore the delay of the approximation up to the next period of the top-most parent element requires at most k additional test intervals for the child. The value $2k$ is therefore the bound for the total number of test intervals required for the child. □

Each element can require as many splitting points as its total child-set has members. The total child-set contains its children and its children's children.

For reason of simplification we consider only normalized hierarchical event spectra, so spectra in which each hierarchical event element can only have one direct child element at most.

8.2.7. Description model for the real-time calculus. For an exact description for the real-time calculus as proposed in [131] the real-time calculus curves are modeled by an initial non-periodic and a periodic part. Each part is modeled by a set of consecutive piecewise linear curve segments. Each curve segment w is given by the coordinates x,y of its start point and a slope s. Starting the slope at the starting point leads to the coordinates x',y' of the next following segment w'. The periodic part is described by its starting coordinates x_p,y_p, again by a set of piecewise linear curve segments with relative coordinates x,y to the starting point, and by an offset $\Delta x, \Delta y$ between two periods.

DEFINITION 8.2.11. *(similar to [131]) A piecewise linear curve segment $S = (x,y,s)$ is given by the coordinates x,y of its start point and a slope s and specifies a straight line starting from the coordinates x,y with a slope s.*

Each curve of the real-time calculus can be described by such sets.

DEFINITION 8.2.12. *(similar to [131]) An arrival or service curve of the real-time calculus can be described by* $v = \{A, P, p_x, p_y, a_x, a_y\}$ *where A and P are two sets of piecewise linear curve segments, a periodic part* $P = \{S\}$ *and an aperiodic part* $A = \{S\}$*. The offset* a_x, a_y *gives the start of the periodic part. Each segment of A has to have smaller values than* a_x, a_y*. P gives a set of segments describing the periodic part of the curve. Its coordinates are given relative to the offset, so the element i occurs at* $a_x + x_i, a_y + y_i$*. The periodic segment is repeated after* p_x*,* p_y*.*

On the one side the curve can be transformed into a special event spectrum. On the other side adding y-values can complete the description of the event spectrum. Then this description is directly a special case of the event spectrum description.

The problem is that each operation on two or more incoming curves requires equalizing the periodic and aperiodic part of the curves first to enable the calculation of the outgoing curves. This equalization requires using the hyper-period of the periods of the incoming curves as the new period. This hyper-period can become quite large and with it the number of segments necessary to describe the curves. To prevent this possible grow of the number of segments it is necessary to bound this number independently of the concrete parameter of the incoming event sequences.

8.2.8. Combined description for the curves. To overcome the problem of hyper-periods we propose to combine the concrete description model of the real-time calculus with the concept of approximation proposed in this thesis. Therefore we propose to use again the idea of approximation and the event spectrum model and to generate a new flat description out of approximated event spectra. The approximated event stream and event spectra lead to a finite set of segment elements with no period part necessary any more.

8.3. Summarizing Examples

EXAMPLE 8.3.1. Fig. 8.3.1 (example 8.1.2) shows the advanced approximation for the spectrum bound function of the event spectrum:

$$\hat{\Theta}_7 = \{(20\,ms, 0\,ms, 10\,ms, 0\,\frac{s}{s}, (2\,ms, 0\,ms, 2\,ms, \infty\,\frac{s}{s}, \emptyset))\}$$

The event spectrum consists of bursts with five events. The advanced approximated event spectrum with an approximation after three events has the following separation points:

$$s_{1,0} = 0\,ms$$
$$s_{1,1} = 20\,ms$$
$$s_{2,0} = 0\,ms$$
$$s_{2,1} = 4\,ms$$
$$s_{2,2} = 60\,ms$$

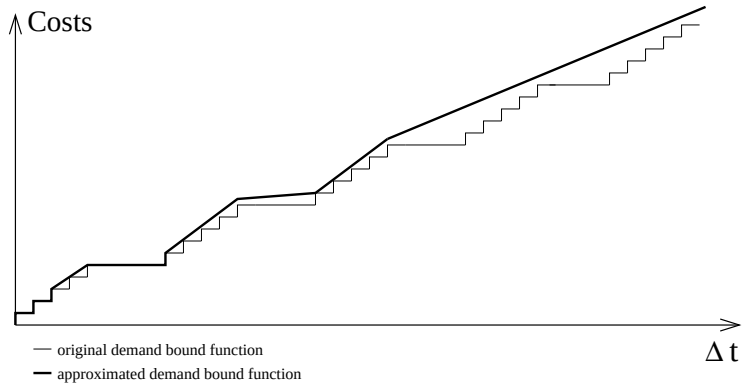

FIGURE 8.3.1. Example 8.1.2: Approximated hierarchical event bound function

For Δx and Δy we have the values:

$$\Delta x = \frac{L - L'}{\frac{L'}{p'}} + a' = \frac{10\,ms - 2\,ms}{\frac{2\,ms}{2\,ms}} + 0\,ms = 8\,ms$$

$$\Delta y = L\left(1 - \frac{\Delta x}{p}\right) = 10\,ms\left(1 - \frac{8\,ms}{20\,ms}\right) = 6\,ms$$

Therefore the approximated event spectrum has following description:

$$\begin{aligned}\hat{\Theta}_7^3 =& \{(\infty s, 0\,ms, 10\,ms, 0\frac{s}{s}, \{(20\,ms, 0\,ms, 10\,ms, 0\frac{s}{s}, (2\,ms, 0\,ms, 2\,ms, \infty\frac{s}{s}, \emptyset)),\\ & (\infty s, 10\,ms, 4\,ms, 1\frac{s}{s}, \emptyset)\}), (\infty s, 20\,ms, 12\,ms, 0\frac{s}{s}, \{(\infty s, 0\,ms, 2\,ms, \infty\frac{s}{s}, \emptyset)\\ & , (\infty s, 0\,ms, \infty s, \frac{2\,ms}{20\,ms}, \emptyset), (20\,ms, 0\,ms, 8\,ms, 1\frac{s}{s}, \emptyset)\}),\\ & (\infty s, 60\,ms, 6\,ms, \infty\frac{s}{s}, \emptyset), (\infty s, 60\,ms, \infty s, \frac{1s}{2s}, \emptyset)\}\end{aligned}$$

The result is visualized in figure 8.3.1. The exact event spectrum element is replaced by an approximative event spectrum $\hat{\Theta}_7^3$ consisting of five event spectrum elements $\hat{\theta}_{7,1}^3, \hat{\theta}_{7,2}^3, \hat{\theta}_{7,3}^3, \hat{\theta}_{7,4}^3$. The first event spectrum element

$$\begin{aligned}\hat{\theta}_{7,1}^3 =& (\infty s, 0\,ms, 10\,ms, 0\frac{s}{s}, \{(20\,ms, 0\,ms, 10\,ms, 0\frac{s}{s}, (2\,ms, 0\,ms, 2\,ms, \infty\frac{s}{s}, \emptyset)),\\ & (\infty s, 10\,ms, 4\,ms, 1\frac{s}{s}, \emptyset)\})\end{aligned}$$

describes the non-approximated part. It ends somewhere within the first period of $\hat{\theta}_7^3$. The second child spectrum element of $\hat{\theta}_{7,1}^3$, the spectrum element $\hat{\theta}_{7,1,2}^3 = (\infty s, 10\,ms, 4\,ms, 1\frac{s}{s}, \emptyset)$ describes the start of the approximation of $\hat{\theta}_7$ for the remaining events of the first period of $\hat{\theta}_7$.

The second spectrum element $\hat{\theta}_{7.2}^3$ describes the part in which the child spectrum element is approximated but the parent spectrum element is not approximated. It has three child spectrum elements $\hat{\theta}_{7,2,1}^3, \hat{\theta}_{7,2,2}^3$ and $\hat{\theta}_{7,2,3}^3$ where the first child spectrum element $\hat{\theta}_{7,2,1}^3 = (\infty s, 0\,ms, 2\,ms, \infty\frac{s}{s}, \emptyset)$ describes the initial event of the embedded event spectrum element once to allow the approximation of this initial event by the second child spectrum

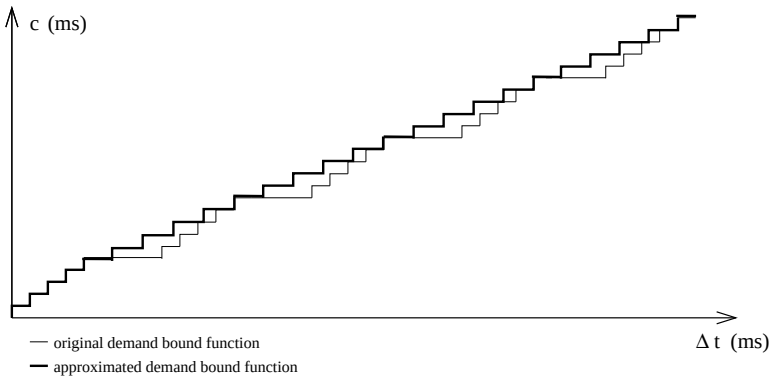

FIGURE 8.3.2. Example 8.1.2: Periodic model with minimum separation distance

element $\hat{\theta}^3_{7,2,2} = (\infty s, 0\,ms, \infty s, \frac{2s}{20s}, \emptyset)$. These first events of the child spectrum element are considered separately from the remaining events of the child spectrum element. They are approximated using the period of the parent spectrum element $\hat{\theta}_7$. This separate approximation of the first child-events guarantees that the spectrum bound function of the approximated spectrum element $\hat{\theta}^3_7$ meets or exceeds the spectrum bound function of the original element $\hat{\theta}_7$ at every first event of the child element.

The remaining events of the child spectrum element can than be approximated by a slope starting at these first events. $\hat{\theta}^3_{7,2,2}$ is the approximation for every first event of the child spectrum element. The element $\hat{\theta}^3_{7,2,3} = (20\,ms, 0\,ms, 8\,ms, 1\frac{s}{s}, \emptyset))$ models the approximation of the rest of the child spectrum element. Note that, as the approximation of every first event is already done by the elements $\hat{\theta}^3_{7,2,1}$ and $\hat{\theta}^3_{7,2,2}$, the approximation of the child spectrum element has to include one event less than the original child spectrum element.

The third and the fourth element of the approximated parent spectrum describe finally the approximation of the parent spectrum element. The third spectrum element $\hat{\Theta}^3_{7,3} = (\infty s, 60\,ms, 6\,ms, \infty\frac{s}{s}, \emptyset)$ gives the cost offset necessary to start the approximation at the right cost level. The fourth spectrum element $\hat{\theta}^3_{7,4} = (\infty s, 60\,ms, \infty ms, \frac{1s}{2s}, \emptyset)$ models the part in which also the parent spectrum element is approximated. This spectrum element is independent of the child spectrum element; only the period $p_{\hat{\theta}_7}$ and the limitation $L_{\hat{\theta}_7}$ of the parent spectrum element $\hat{\theta}_7$ are required to calculate the approximating slope.

A characteristic of such a description is the separate limitation of the maximum number of test intervals for each spectrum element. In the example five test intervals for the child element and four test intervals for the parent element are required.

To demonstrate the advantages of the new model and the approximation we give the description of $\hat{\Theta}_7$ for the periodic model with minimum separation distance (figure 8.3.2), for the exact and for the approximative description of the real-time calculus (figure 8.3.3).

The nearest description of $\hat{\Theta}_7$ with the periodic model with minimum separation distance is period $p = 4\,ms$, a minimum separation distance $s = 2\,ms$, a worst-case execution time of $c^+ = 2\,ms$ and a jitter $j = 8\,ms$. The first burst is modeled exactly by this

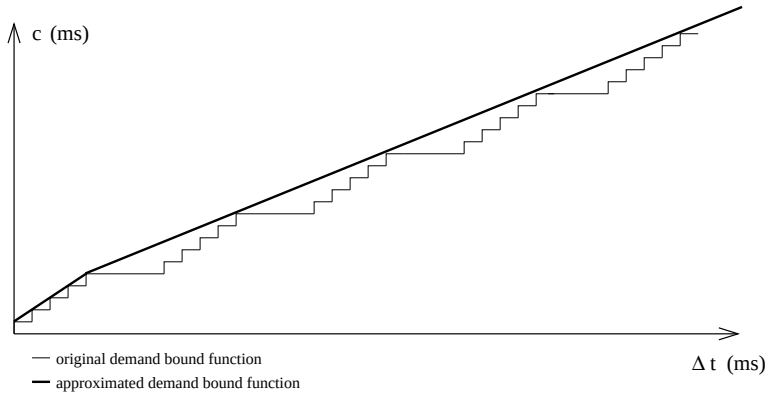

FIGURE 8.3.3. Example 8.1.2: Approximation of the real-time calculus

description. The remaining events are approximated by a periodic function. The worst absolute approximation error occurs for example at interval $\Delta t = 16\,ms$ and has the value $c = \left(\left\lfloor \frac{16\,ms+8\,ms}{4\,ms} + 1 \right\rfloor 2\,ms\right) - 10\,ms = 4\,ms$. The worst relative approximation error occurs for the same interval $\Delta t = 16\,ms$ and is $\frac{4\,ms}{10\,ms} = 40\%$.

Despite that the overestimation of the periodic model is up to 40% of the real demand, the number of test intervals required in the worst case by the periodic model is not bounded. The model can produce an infinitely set of test intervals. So, for the analysis, other bounds as a busy period or a good fixed-point iteration are required.

For the approximation of the real-time calculus [39] each curve is described by three linear segments w_1, w_2, w_3 with $w = (x, y, s)$ where x, y are the coordinates of the start point of the segment and s is the slope of the segment. For the example the description would by $w_1 = (0\,ms, 0\,ms, \infty \frac{s}{s})$, $w_2 = (0\,ms, 2\,ms, 1\frac{s}{s})$ and $w_3 = (8\,ms, 10\,ms, \frac{1s}{2s})$. So we only have three test intervals for the analysis. The maximum absolute approximation error for this model occurs directly before interval $\Delta t = 20\,ms$ so at the interval $\Delta t = 19,999999...\,ms$. The size of the error is nearly $c = 10\,ms + \frac{12\,ms}{\frac{2s}{1s}} - 10\,ms = 6\,ms$ leading to a relative error of $\frac{6\,ms}{10\,ms} = 60\%$. The worst case relative error occurs unfortunately very early just before interval $\Delta t = 2\,ms$ with a relative error of $\frac{2\,ms}{2\,ms} = 100\%$.

Due to this inexactness an exact description is used for the real-time calculus [131]. It was introduced in chapter 2.3.6. The curve is described by an aperiodic and a periodic part, both containing a set of piecewise linear segments. For the example event spectrum $\hat{\Theta}_7$ only the periodic part is required with the segments $w_1 = (0\,ms, 2\,ms, 0\frac{s}{s})$, $w_2 = (2\,ms, 2\,ms, 0\frac{s}{s})$, $w_3 = (4\,ms, 4\,ms, 0\frac{s}{s})$, $w_4 = (6\,ms, 6\,ms, 0\frac{s}{s})$ and $w_5 = (8\,ms, 8\,ms, 0\frac{s}{s})$. The segments are repeated with an an offset $\Delta x = 20\,ms$ and $\Delta y = 10\,ms$. This description models the event spectrum $\hat{\Theta}_7$ exactly. But the number of test intervals required for schedulability analysis can become quite high and the description suffers from the same problems as the description of event streams (see 2.3.2). A burst with more events would require one additional segment for each additional event within the burst, so a burst of 1000 events would require for example 1000 segments.

The error for the approximation of the hierarchical event spectrum is bounded by the chosen degree of exactness. In this example with an approximation after 3 events the error is 33%, choosing an approximation after 10 test intervals reduces the error to 10% and 1% for an approximation after 100 events. The required number of test intervals is bounded and is 8 test intervals for the approximation after 3 events and about 30 test intervals in the case of 10 events and not more than 3000 test intervals after 100 events.

EXAMPLE 8.3.2. Let us reconsider example 8.1.3. We have an event spectrum element $\hat{\theta}_8 = \{1000\,ms, 0\,ms, 150\,ms, 0\frac{s}{s}, \{\hat{\theta}_{8,b}\}\}$ with a child event spectrum element $\hat{\theta}_{8,b} = \{10\,ms, 0\,ms, 5\,ms, \infty\frac{s}{s}, \emptyset\}$. The approximation is set to 100 test intervals, that means it starts within the 4-th period of the $\hat{\theta}_8$.

We have:

$$\Delta x = \frac{L_{\hat{\theta}_8} - L_{\hat{\theta}_{8,b}}}{\frac{L_{\hat{\theta}_{8,b}}}{p_{\hat{\theta}_{8,b}}}} - a_{\hat{\theta}_{8,b}} = \frac{150\,ms - 5\,ms}{\frac{5\,ms}{10\,ms}} - 0\,ms = 290\,ms$$

$$\Delta y = L_{\hat{\theta}_8}\left(1 - \frac{\Delta x}{p_{\hat{\theta}_8}}\right) = 150\,ms\left(1 - \frac{290\,ms}{1000\,ms}\right) = 106.5\,ms$$

For the example the resulting approximated event spectrum $\hat{\theta}_8^{100}$ reads as follows:

$$\begin{aligned}\hat{\theta}_8^{100} =& \{(\infty s, 0\,ms, 600\,ms, 0\frac{s}{s}, \hat{\theta}_8),\\ & (\infty s, 4000\,ms, 14400\,ms, 0\frac{s}{s}, \{(\infty s, 4000\,ms, 5\,ms, \infty\frac{s}{s}, \emptyset),\\ & (\infty s, 4000\,ms, \infty s, \frac{5\,ms}{1000\,ms}, \emptyset), (1000\,ms, 0\,ms, 145\,ms, \frac{5\,ms}{10\,ms}, \emptyset)\}),\\ & (\infty, 100\,s, 106.5\,ms, \infty\frac{s}{s}, \emptyset), (\infty s, 100\,s, \infty, \frac{150\,ms}{1000\,ms}, \emptyset)\}\end{aligned}$$

First the exact description is used for the first four periods of $\hat{\theta}_8$. Then the approximation of the child spectrum element $\hat{\theta}_{8,b}$ starts. Each first event of the child spectrum is approximated by $\hat{\theta}_{8,2,1}^{100} = (\infty\ s, 4\,s, 5\,ms, \infty\frac{s}{s}, \emptyset)$ and $\hat{\theta}_{8,2,2}^{100} = (\infty s, 4\,s, \infty s, \frac{5\,ms}{1000\,ms}, \emptyset)$, the remaining events are approximated separately for each period of $\hat{\theta}_8$ with

$\hat{\theta}_{8,2,3}^{100} = (1\,s, 0\,ms, 145\,ms, \frac{5\,ms}{10\,ms}, \emptyset)\})$. The next two spectrum elements $\hat{\theta}_{8,3}^{100}$ and $\hat{\theta}_{8,4}^{100}$ model the approximation of the parent spectrum element $\hat{\theta}_8$.

In the periodic model with minimum separation distance the example would be approximated by the period $p = \frac{1s}{30}$, the minimum separation distance $s = 10\,ms$, a jitter $j = 1000\,ms - \frac{1000\,ms}{30} - 270\,ms$ and the worst-case execution time $c^+ = 5\,ms$. The maximum absolute error occurs just before interval $\Delta t = 1\,s$ and has a size of $120\,ms$ resulting in a maximum relative error of $\frac{120}{150} = 80\%$.

The approximation of the real-time calculus has for this example the segments $w_1 = (0\,ms, 0\,ms, \infty\frac{s}{s})$, $w_2 = (0\,ms, 5\,ms, \frac{1s}{2s})$ and $w_3 = (270\,ms, 150\,ms, \frac{150\,ms}{1000\,ms})$ resulting in a relative error of $120\,ms$ and an absolute error of 73%.

The exact description of the real-time calculus has an error of 0%. For this description only the periodic part is required with the segments

$$w_1 = (0\,ms, 5\,ms, 0\frac{s}{s}), w_2 = (10\,ms, 10\,ms, 0\frac{s}{s}), w_3 = (20\,ms, 15\,ms, 0\frac{s}{s})$$

$$w_4 = (30\,ms, 20\,ms, 0\frac{s}{s}), ..., w_{30} = (270\,ms, 150\,ms, 0\frac{s}{s})$$

and the periodic offsets $\Delta x = 1000\,ms$ and $\Delta y = 150\,ms$. Its is obvious how the description size depends on the length of bursts.

The approximation of the hierarchical event spectra has an error of 1% and a number of considered test intervals of $300\,ms$.

CHAPTER 9

Case-Study

In order to show the generality of the new event spectra model we will conduct a case study. The system we explore has been published in [77] and is depicted in figure 9.0.1 (all times in ms). In the paper the system has been analyzed by two different compositional analysis methods. The first one is the Modular Performance Analysis (MPA) implementing the real-time calculus [131] and the other one the extended periodic model [117]. The two approaches had been combined in order to get tighter bounds in the real-time analysis [77]. We will show that we are able to conduct an analysis with our model and get tighter bounds as well. However, to show how tight the approach is in general we have simulated the task set, too.

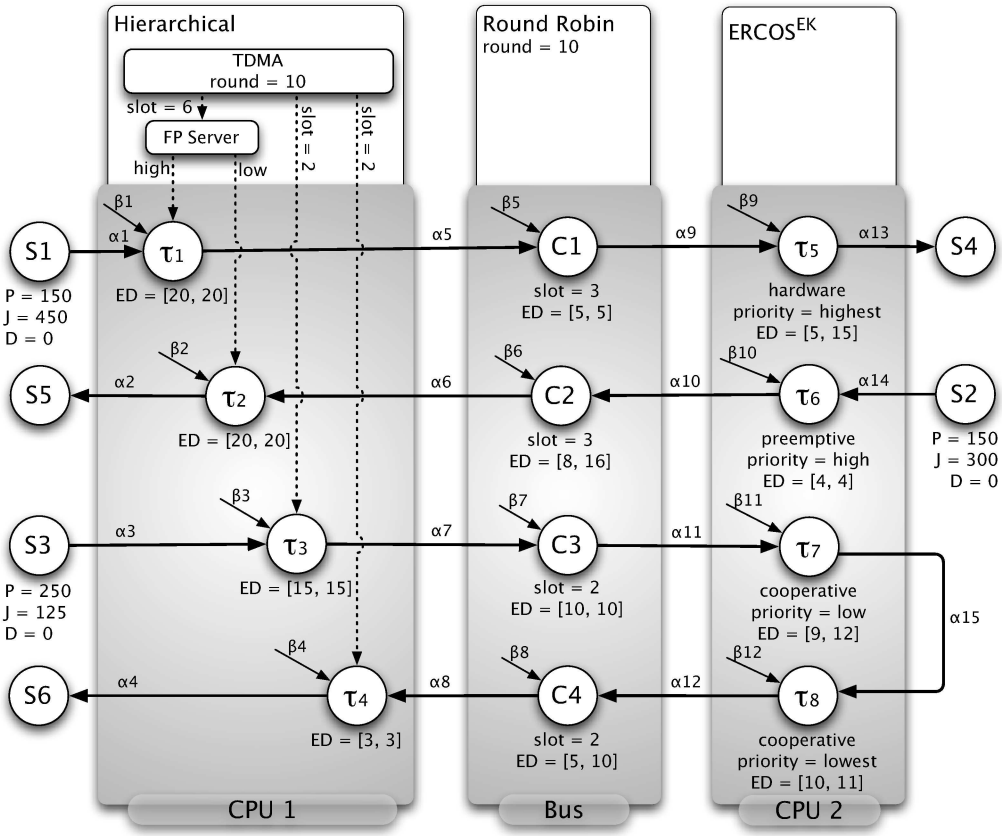

FIGURE 9.0.1. Example of a distributed hard-real time system published in [77]

The system consists of two CPUs connected by a bus. CPU 1 executes four tasks scheduled by a hierarchical scheduler. At the top-level a TDMA policy is implemented. The tasks τ_1 and τ_2 have got 60% of the TDMA cycle available. Within this slot a fixed-priority scheduler schedules both tasks. The remaining tasks τ_3 and τ_4 have each 20% of the TDMA cycle available. The bus uses a round robin scheduler for the communication. Four tasks with different time slots are provided. CPU 2 executes also four tasks. The tasks are scheduled by the priority-based scheduler of the $ERCOS^{EK}$ operating system. The special point which has to be considered in this example is the cooperative behavior of the tasks τ_7 and τ_8. The task τ_7 has got a higher priority than τ_8 but when τ_8 is executed the task τ_7 can only interrupt the task τ_8 at specific points in time. These points are 2, 3, 4, 6 or 11 ms.

Three paths are given in this distributed system: $S1 \rightarrow S4$, $S2 \rightarrow S5$ and $S3 \rightarrow S6$. The first path $S1 \rightarrow S4$ consists of the tasks τ_1, c_1 and τ_5. Task τ_1 activates over the communication task c_1 the task τ_5. The second path $S2 \rightarrow S5$ starts on CPU 2. The task τ_6 activates over the communication task c_2 the task τ_2. The last path $S3 \rightarrow S6$ begins with the task τ_3 activating the communication task c_3 that then activates the task τ_7. The task τ_7 activates the cooperative task τ_8. Finally, the task τ_8 activates via the communication task c_4 the task τ_4 on the CPU 1.

For all three paths hard real-time constraints are given. The first path $S1 \rightarrow S4$ must not exceed 200 ms. The two other paths must not exceed 400 ms. The aim of the analysis is to calculate for each path its latency as accurately as possible. To solve the problem we have to determine all the necessary event spectra and service spectra in the system. We will describe only those spectra of the system that are necessary for the latency of the three paths. Thereby we choose different domains for the illustration of the event spectra and service spectra. The event spectra describe for each interval the number of events and the service spectra describe for each interval the available execution time. This illustration is better to read, but for the calculation we have to transform the descriptions into a common domain. This means, for example, we scale the event spectra with the execution times.

We will explore the latency of the path $S1 \rightarrow S4$ first. This is necessary, because the results have a direct influence of the other paths. The results of the spectra of path one are represented in Table 1.

τ_1	[ms]
α_1^u	$\{(\infty,0,4,\infty,\emptyset),(150,150,1,\infty,\emptyset)\}$
α_1^l	$\{(150,600,1,\infty,\emptyset)\}$
β_1^u	$\{(10,0,6,1,\emptyset)\}$
β_1^l	$\{(10,4,6,1,\emptyset)\}$
α_5^u	$\{(\infty,0,3,0,\{(32,0,1,\infty,\emptyset)\}),(\infty,100,1,\infty,\emptyset),(150,150,1,\infty,\emptyset)\}$
α_5^l	$\{(150,604,1,\infty,\emptyset)\}$
β_2^u	$\{(\infty,0,340,0,\{(10,0,6,1,\emptyset)\}),(150,600,70,0,\{10,0,6,1,\emptyset\})\}$
β_2^l	$\{(\infty,136,4,1,\emptyset),(\infty,144,6,1,\emptyset),(150,186,4,1,\emptyset),$ $(150,194,66,0,\{(10,0,6,1,\emptyset)\})\}$

TABLE 1. Results of the computed event spectra of the path $S1 \rightarrow S4$ (I)

c_1	[ms]
α_5^u	$\{(\infty,0,3,0,\{(32,0,1,\infty,\emptyset)\}),(\infty,100,1,\infty,\emptyset),(150,150,1,\infty,\emptyset)\}$
α_5^l	$\{(150,604,1,\infty,\emptyset)\}$
β_5^u	$\{(\infty,0,\infty,1,\emptyset)\}$
β_5^l	$\{(10,7,3,1,\emptyset)\}$
α_9^u	$\{(\infty,0,1,\infty,\emptyset),(\infty,18,2,0,\{(32,0,1,\infty,\emptyset)\}),$ $(\infty,86,1,\infty,\emptyset),(150,136,1,\infty,\emptyset)\}$
α_9^l	$\{(150,618,1,\infty,\emptyset)\}$

TABLE 2. Results of the computed event spectra of the path $S1 \rightarrow S4$ (II)

τ_5	[ms]
α_9^u	$\{(\infty,0,1,\infty,\emptyset),(\infty,18,2,0,\{(32,0,1,\infty,\emptyset)\}),$ $(\infty,86,1,\infty,\emptyset),(150,136,1,\infty,\emptyset)\}$
α_9^l	$\{(150,618,1,\infty,\emptyset)\}$
β_9^u	$\{(\infty,0,\infty,1,\emptyset)\}$
β_9^l	$\{(\infty,0,\infty,1,\emptyset)\}$
β_{10}^u	$\{(\infty,0,613,1,\emptyset),(150,618,145,1,\emptyset)\}$
β_{10}^l	$\{(\infty,15,3,1,\emptyset),(\infty,33,17,1,\emptyset),(\infty,65,21,1,\emptyset),$ $(\infty,101,35,1,\emptyset),(150,151,135,1,\emptyset)\}$

TABLE 3. Results of the computed event spectra of the path $S1 \rightarrow S4$ (III)

We start the calculation with the task τ_1 on CPU 1. The task has got the highest priority within its TDMA-slot. So the completely available capacity of β_1 can be used. According to chapter 7.6 we can determine the event spectrum of α_5. Additionally, we have to determine the remaining capacity for task τ_2, so that we can determine the latency of the path $S2 \rightarrow S5$.

The next step is to determine the event spectrum of α_9. To do this we have to determine the capacity of β_5 first. The upper bound for a round robin policy can be simply approximated by the full processor capacity, because the best case occurs when all other tasks (c_2, c_3, c_4) have no jobs to execute. The lower bound, in this case, must be approximated by a TDMA policy. The worst case for a task in a round robin scheduler occurs when all other tasks use in every cycle their full slot time. Obviously, this is equal to a TDMA policy. Since we do not know the arrival curves of the tasks c_2, c_3 and c_4, we have to assume that the bus is completely busy.

The last task in the chain is τ_5. This task has the full processor capacity and is activated by α_9. The outgoing event spectrum α_{13} is not necessary for the path latency. So the only spectrum we have to determine is the remaining capacity for the task τ_6, so that we are able to calculate the other paths. Finally, we determine the end-to-end deadline. According to the theorem in chapter 7.7.3 about the end-to-end response-times the resulting latency of the first path is 170 ms. The task τ_1 has a worst case response time of 136 ms, c_1 of 19 ms and τ_5 of 15 ms.

The second path we will explore is the path $S2 \rightarrow S5$. The results of this path are equally important for the analysis of the last path as the first path. The results are represented in Table 4. Again we have only calculated the spectra necessary for the path latency.

τ_6	[ms]
α_{14}^u	$\{(\infty,0,3,\infty,\emptyset),(150,150,1,\infty,\emptyset)\}$
α_{14}^l	$\{(150,450,1,\infty,\emptyset)\}$
β_{10}^u	$\{(\infty,0,613,1,\emptyset),(150,618,145,1,\emptyset)\}$
β_{10}^l	$\{(\infty,15,3,1,\emptyset),(\infty,33,17,1,\emptyset),(\infty,65,21,1,\emptyset),(\infty,101,35,1,\emptyset),$ $(150,151,135,1,\emptyset)\}$
α_{10}^u	$\{(\infty,0,3,\infty,\{(4,0,1,\infty,\emptyset)\},(150,121,1,\infty,\emptyset)\}$
α_{10}^l	$\{(150,480,1,\infty,\emptyset)\}$
β_{11}^u	$\{(\infty,0,450,1,\emptyset),(150,455,145,\emptyset)\}$
β_{11}^l	$\{(\infty,42,8,1,\emptyset),(\infty,65,21,1,\emptyset),(\infty,101,35,1,\emptyset),(150,151,131,1,\emptyset)\}$

TABLE 4. Event spectra of the path $S2 \rightarrow S5$ task τ_6

c_2	[ms]
α_{10}^u	$\{(\infty,0,3,\infty,\{(4,0,1,\infty,\emptyset)\},(150,121,1,\infty,\emptyset)\}$
α_{10}^l	$\{(150,480,1,\infty,\emptyset)\}$
β_6^u	$\{(\infty,0,\infty,1,\emptyset)\}$
β_6^l	$\{(\infty,7,12,0,\{(33,0,3,1,\emptyset),(33,9,3,1,\emptyset),(33,16,3,1,\emptyset),(33,23,3,1,\emptyset)\}),$ $(\infty,106,3,1,\emptyset),(\infty,115,3,1,\emptyset),(\infty,122,12,0,\{(7,0,3,1,\emptyset)\}),(150,153,3,1,\emptyset),$ $(150,162,3,1,\emptyset),(150,169,57,0,\{(7,0,3,1,\emptyset)\}\}$
α_6^u	$\{(\infty,0,3,0,\{(8,0,1,\infty,\emptyset)\}),(150,70,1,\infty,\emptyset)\}$
α_6^l	$\{(150,523,1,\infty,\emptyset)\}$

TABLE 5. Event spectra of the path $S2 \rightarrow S5$ task C_2

τ_2	[ms]
α_6^u	$\{(\infty,0,3,0,\{(8,0,1,\infty,\emptyset)\}),(150,70,1,\infty,\emptyset)\}$
α_6^l	$\{(150,523,1,\infty,\emptyset)\}$
β_2^u	$\{(\infty,0,340,0,\{(10,0,6,1,\emptyset)\}),(150,600,70,0,\{10,0,6,1,\emptyset\})\}$
β_2^l	$\{(\infty,136,4,1,\emptyset),(\infty,144,6,1,\emptyset),(150,186,4,1,\emptyset),$ $(150,194,66,0,\{(10,0,6,1,\emptyset)\})\}$

TABLE 6. Event spectra of the path $S2 \rightarrow S5$ task τ_2

The incoming event spectrum α_{14} of τ_6 on CPU 2 is given and the remaining capacity β_{10} has been calculated during the first path calculation. So we can straightforward determine the event spectrum α_{10}. Furthermore, we have to determine the remaining capacity β_{11} which will be used during the third path calculation.

To calculate the event spectrum α_6 we have to determine the capacity spectrum β_6 first. The upper bound of β_6 is the full capacity of the processor like α_5^u. The lower bound of β_6 of the capacity can be approximated by a TDMA policy. Since the arrival curve α_5 is known, we are able to include the real round robin policy at this point and not only the TDMA-approximation. This leads to more capacity compared to β_5, because the task c_1 is not busy in every cycle. Consequently, this leads to a more relaxed α_6.

For the last task τ_2 we have all information in order to determine the maximum latency. The remaining capacity has no influence of the last path $S3 \rightarrow S6$.

We are able to determine the end-to-end deadline of the path $S5 \rightarrow S2$. According to the theorem in chapter 7.7.3 about the end-to-end response time the resulting latency

of the second path $S5 \rightarrow S2$ is 366 ms. In the example it is the result of the worst-case response time of the first job of τ_6, the third job of c_2 and again the first job of τ_2. The reason to take the third job of c_2 is that response-times for c_2 are much longer than for τ_6 and therefore the second and third job of τ_6 arrive before they are required by c_2. The worst-case occurs when τ_2 experiences its worst-case delay exactly at the time when c_2 finishes the execution of its third job. The calculation of the other end-to-end response times follows a corresponding scheme. The task τ_6 has a worst-case response time of 34 ms, c_2 of 132 ms and τ_2 of 200 ms.

τ_3	[ms]
α_3^u	$\{(\infty,0,1,\infty,\emptyset),(250,125,1,\infty,\emptyset)\}$
α_3^l	$\{(250,375,1,\infty,\emptyset)\}$
β_3^u	$\{(10,0,2,1,\emptyset)\}$
β_3^l	$\{(10,8,2,1,\emptyset)\}$
α_7^u	$\{(\infty,0,1,\infty,\emptyset),(250,117,1,\infty,\emptyset)\}$
α_7^l	$\{(250,383,1,\infty,\emptyset)\}$

TABLE 7. Event spectra of the path $S3 \rightarrow S6$ task τ_3

The last path we have to explore is $S3 \rightarrow S6$. The event spectra are given in Table 7 and Table 10. We start with the task τ_3. Here we have only to determine the outgoing spectrum

c_3	[ms]
α_7^u	$\{(\infty,0,1,\infty,\emptyset),(250,117,1,\infty,\emptyset)\}$
α_7^l	$\{(250,383,1,\infty,\emptyset)\}$
β_7^u	$\{(\infty,0,\infty,1,\emptyset)\}$
β_7^l	$\{(\infty,8,12,0,\{(33,0,2,1,\emptyset),(33,9,2,1,\emptyset),(33,16,2,1,\emptyset),(33,23,2,1,\emptyset)\}),$ $(\infty,107,2,1,\emptyset),(\infty,116,2,1,\emptyset),(\infty,123,8,0,\{(7,0,2,1,\emptyset)\}),$ $(\infty,154,2,1,\emptyset),(\infty,163,2,1,\emptyset),(\infty,170,2,1,\emptyset),(\infty,175,2,1,\emptyset),$ $(\infty,181,46,1,0,\{(4,0,2,1,\emptyset)\})(150,274,10,0,\{(7,0,2,1,\emptyset)\}),$
α_{11}^u	$\{(\infty,0,1,\infty,\emptyset),(250,82,1,\infty,\emptyset)\}$
α_{11}^l	$\{(250,418,1,\infty,\emptyset)\}$

TABLE 8. Event spectra of the path $S3 \rightarrow S6$ task C_3

τ_7	[ms]
α_{11}^u	$\{(\infty,0,1,\infty,\emptyset),(250,82,1,\infty,\emptyset)\}$
α_{11}^l	$\{(250,418,1,\infty,\emptyset)\}$
β_{11}^u	$\{(\infty,0,450,1,\emptyset),(150,455,145,\emptyset)\}$
β_{11}^l	$\{(\infty,42,8,1,\emptyset),(\infty,65,21,1,\emptyset),(\infty,101,35,1,\emptyset),(150,151,131,1,\emptyset)\}$
α_{15}^u	$\{(\infty,0,1,\infty,\emptyset),(250,22,1,\infty,\emptyset)\}$
α_{15}^l	$\{(250,478,1,\infty,\emptyset)\}$
β_{12}^u	$\{(\infty,0,418,1,\emptyset),\{(\infty,427,23,1,\emptyset),(750,455,145,1,\emptyset),$ $(750,605,63,1,\emptyset),(750,677,73,1,\emptyset),(750,755,145,1,\emptyset),$
β_{12}^l	$\{(\infty,69,13,1,\emptyset),(\infty,109,27,1,\emptyset),(750,151,131,1,\emptyset),(750,301,31,1,\emptyset),$ $(750,344,88,1,\emptyset),(750,451,31,1,\emptyset),(750,594,88,1,\emptyset),(750,601,131,1,\emptyset),$ $(750,601,131,1,\emptyset),(750,751,31,1,\emptyset),(750,744,88,1,\emptyset)\}$

TABLE 9. Event spectra of the path $S3 \rightarrow S6$ task τ_7

τ_8	[ms]
α_{15}^u	$\{(\infty,0,1,\infty,\emptyset),(250,22,1,\infty,\emptyset)\}$
α_{15}^l	$\{(250,475,1,\infty,\emptyset)\}$
β_{12}^u	$\{(\infty,0,418,1,\emptyset),\{(\infty,427,23,1,\emptyset),(750,455,145,1,\emptyset),(750,605,63,1,\emptyset),$ $(750,677,73,1,\emptyset),(750,755,145,1,\emptyset),(750,905,13,1,\emptyset),(750,927,123,1,\emptyset),$ $(750,1055,113,1,\emptyset),(750,1177,23,1,\emptyset)\}$
β_{12}^l	$\{(\infty,69,13,1,\emptyset),(\infty,109,27,1,\emptyset),(750,151,131,1,\emptyset),(750,301,31,1,\emptyset),$ $(750,344,88,1,\emptyset),(750,451,31,1,\emptyset),(750,594,88,1,\emptyset),(750,601,131,1,\emptyset),$ $(750,601,131,1,\emptyset),(750,751,31,1,\emptyset),(750,744,88,1,\emptyset)\}$
α_{12}^u	$\{(\infty,0,1,\infty,\emptyset),(\infty,11,1,\infty,\emptyset),(250,216,1,\infty,\emptyset)\}$
α_{12}^l	$\{(\infty,0,531,1,\emptyset),(250,781,1,\emptyset)\}$

TABLE 10. Event spectra of the path $S3 \rightarrow S6$ task τ_8

c_4	[ms]
α_{12}^u	$\{(\infty,0,1,\infty,\emptyset),(\infty,11,1,\infty,\emptyset),(250,216,1,\infty,\emptyset)\}$
α_{12}^l	$\{(\infty,0,531,1,\emptyset),(250,781,1,\emptyset)\}$
β_8^u	$\{(\infty,0,\infty,1,\emptyset)\}$
β_8^l	$\{(\infty,8,2,1,\emptyset),(\infty,17,2,1,\emptyset),(\infty,24,2,1,\emptyset),(\infty,31,2,1,\emptyset),(\infty,41,2,1,\emptyset),$ $(\infty,48,2,1,\emptyset),(\infty,53,2,1,\emptyset),(\infty,58,2,1,\emptyset),(\infty,63,2,1,\emptyset),(\infty,81,2,1,\emptyset),$ $(\infty,88,2,1,\emptyset),(\infty,93,2,1,\emptyset),(\infty,98,2,1,\emptyset),(\infty,106,2,1,\emptyset),$ $(\infty,113,14,0,\{7,0,2,1,\emptyset\}),(\infty,159,2,1,\emptyset),(750,274,10,0,$ $\{(5,0,2,1,\emptyset)\}),(750,297,2,1,\emptyset),(750,299,2,1,\emptyset),(750,304,2,1,\emptyset),$ $(750,308,59,1,\emptyset),(750,424,10,0,\{(5,0,2,1,\emptyset)\}),(750,447,2,1,\emptyset),$ $(750,449,2,1,\emptyset),(750,454,2,1,\emptyset),(750,458,114,1,\emptyset),(750,574,10,0,$ $\{(5,0,2,1,\emptyset)\}),(750,602,2,1,\emptyset),(750,604,2,1,\emptyset),(750,609,2,1,\emptyset),$ $(750,613,4,1,\emptyset),(750,369,10,0,\{(4,0,2,1,\emptyset\}),(750,387,34,1,\emptyset),$ $(750,619,10,0,\{(4,0,2,1,\emptyset\}),(750,637,84,1,\emptyset)\}$
α_8^u	$\{(\infty,0,1,\infty,\emptyset),(\infty,5,1,\infty,\emptyset),(250,176,1,\infty,\emptyset)\}$
α_8^l	$\{(\infty,0,571,1,\emptyset),(250,821,1,\emptyset)\}$

TABLE 11. Event spectra of the path $S3 \rightarrow S6$ task c_4

τ_4	[ms]
α_8^u	$\{(\infty,0,1,\infty,\emptyset),(\infty,5,1,\infty,\emptyset),(250,176,1,\infty,\emptyset)\}$
α_8^l	$\{(\infty,0,571,1,\emptyset),(250,821,1,\emptyset)\}$
β_4^u	$\{(\infty,0,\infty,1,\emptyset)\}$
β_4^l	$\{(\infty,8,\infty,1,\emptyset)\}$

TABLE 12. Event spectra of the path $S3 \rightarrow S6$ task τ_4

α_8. The same applies for the communication task c_3. Here we have only to determine α_{11}. The capacity for the task can be determined like the scheme used for β_6. The upper bound is the full processor capacity. For the lower bound we use the round robin policy for tasks having a calculated arrival curve and for the other one the TDMA policy.

Now we have to consider the cooperative tasks τ_7 and τ_8. Since we have the capacity β_{11} from the second path and the incoming event spectrum, we can straightforwardly determine the remaining capacity and the event spectrum for task τ_8. But at this point we have to consider a further fact. The task τ_8 can only be activated when the task τ_7 has been executed. By means of this we are able to calculate the outgoing event spectrum α_{12}. It

can be seen that task τ_7 cannot influence the task τ_8. So it is not necessary to consider the cooperative behavior.

In order to calculate the results of the communication task c_4 we can use the same policy as for task c_3. When the event spectrum α_8 is calculated, we have all necessary spectra computed. The outputs from task τ_4 are not important for the path latency.

Now we are able to determine the end-to-end worst-case response time. According to the theorem 7.7.3 about the end-to-end response time in chapter 7.7.3 the resulting latency of the second path $S3 \rightarrow S6$ is 321 ms. The task τ_3 has a worst case response time of 71 ms, c_3 of 45 ms, τ_7 of 69 ms, τ_8 of 80 ms, c_4 of 45 ms and τ_4 of 11 ms.

Finally, we compare our results versus the modular performance analysis (MPA) and the extended periodic model and their combination. In order to quantify our exactness we have simulated the example model, too. This simulation has been done with the real-time simulator chronSim [123, 124] and gives a (tight) lower bound on the worst-case response-times. The results of the different approaches are presented in Table 13. ESC means the Event Spectra Calculus.

PATH	MPA	extended Periodic	MPA & extended Periodic	ESC	Constraint	ChronSim
$S1 \rightarrow S4$	170 ms	170 ms	170 ms	170 ms	200 ms	155 ms
$S2 \rightarrow S5$	430 ms	376 ms	376 ms	366 ms	400 ms	242 ms
$S3 \rightarrow S6$	412 ms	422 ms	389 ms	321 ms	400 ms	206 ms

TABLE 13. Results of the MPA, extended Periodic, the combination MPA and extended Periodic, ESC and ChronSim

It can be seen that our approach delivers tighter bounds for the paths $S2 \rightarrow S5$ and $S3 \rightarrow S6$. This is founded by the facts that we only use one model and that we are capable to include dependencies into the analysis. The next table shows the improvement in percent versus extended periodic, MPA and the combination of both.

PATH	MPA	extended periodic	MPA& extended Periodic
$S1 \rightarrow S4$	0%	0%	0%
$S2 \rightarrow S5$	17,4%	2,7%	2,7%
$S3 \rightarrow S6$	28%	31%	21%

TABLE 14. Improvement of the ESC approach in percent versus ext. Periodic, MPA and the combination of both

CHAPTER 10

Summary and Outlook

In this thesis a new integrated and efficient schedulability analysis methodology for uni-processor systems with static and dynamic priorities is proposed.

First some of the existing schedulability approaches were introduced, as there are the approach of Liu & Layland [88], the response time analysis [73], [80] and the scheduling point analysis [92] for static priority systems. For systems with dynamic priorities we have introduced an approach of Liu & Layland [88], the processor demand criterion [19] and the test of Devi [46]. A disadvantage of most of these approaches is the algorithm complexity. The run-time of these algorithms does not only depend on the number of task sets but also on the variables of the concrete task sets and especially on the ration between the smallest and largest tasks in the task sets. Only exceptions are the scheduling point analysis (for non-arbitrary systems), which has an exponential complexity instead and is therefore not suitable for large task sets, and the approaches of Liu & Layland [88] and Devi [46], which are only sufficient or use a limited model. The fixed-priority analysis of Liu & Layland and the EDF analysis of Devi are not usable for systems with utilization larger than 69.3% and 80%, respectively. The EDF analysis of Liu & Layland does only support tasks with simple periodic stimuli and a deadline for each task that is equal or larger than the period of the task's stimuli.

The reason to consider the problem of the run-time of the schedulability analysis is, that the analysis is needed as one step in the design flow of real-time systems. One challenge of the design-flow is to find a good hardware and software design to meet all requirements for the system. For this challenge it is necessary to decide on the hardware-/ software- distribution of the tasks of the system, the allocation of hardware components and their connection network, the partitioning and the binding of the software on the hardware components and the scheduling of the software tasks. To do all these decisions it is necessary to have an optimization-step within the design flow. This step considers a huge number of different candidate systems to find a good one meeting the requirements. A real-time analysis for each of the candidate solutions is required. As the real-time analysis will run quite often, a significant difference in the run-time of the real-time analysis will have a significant impact on the overall performance of the design flow respectively on the number of candidate solutions which can be considered.

The processor demand criterion for the sufficient and necessary analysis of systems with earliest deadline first scheduling has for example a pseudo-polynomial complexity when its utilization is bounded by an upper bound. Its complexity depends on the fraction

between the largest difference between period and deadline of one task in the task set on the one side and the smallest period of a task in the task set on the other side.

To overcome the disadvantage of complexity we have introduced in chapter 3 the concept of approximations and proposed the superposition approximation. It is a fast schedulability analysis with a polynomial run-time being sufficient and nearly necessary. Each system recognized as schedulable is guaranteed to be schedulable. Systems that are not schedulable are therefore also recognized as not schedulable. There might be some systems being schedulable but being not recognized as schedulable by the approximative analysis. The number of such not correctly classified systems depends on a selectable error. We can guarantee that all these systems are quite near to be not schedulable in a sense that they are not schedulable on a processing element with a slightly smaller capacity. Being $\chi(\Delta t)$ the capacity bound function of the original processor and ε be the chosen approximation error we guarantee that these systems are not schedulable on a processor with a capacity of $(1-\varepsilon)\chi(\Delta t)$. The complexity and therefore the run-time of the algorithm depends polynomial on the number of tasks and polynomial on the selected error ε and on nothing else.

Therefore the proposed superposition analysis is a fully polynomial time approximation scheme for the schedulability problem of systems with dynamic priority scheduling.

Also an algorithm was proposed to calculate the minimum necessary capacity for a system to be acceptable by the superposition approximation. This algorithm is quite suitable for on-line analysis for the possibility of using dynamic voltage scaling (DVS).

The superposition algorithm was introduced first for the simply period (sporadic) task model. Then the algorithm was extended to the more advanced event-stream model. This model was introduced by Gresser in [60] and [61] and is quite suitable for modeling general event patterns. It was modified and extended in this work. For example, the concept of periodic event sequences was introduced to give the model a solid theoretical background. A periodic event sequence Θ consists of a set of event elements θ each described by a period p and an offset a. Each element models a set of periodical events having a initial distance of a and each a distance p to their neighboring elements of the same element. A periodic event sequence consists of a set of such elements having a common starting point. We can also interpret the distance as an interval Δt having the common starting point as its start point and the occurrence of the event as the end point of the interval. The event bound function $\eta(\Delta t, \Theta)$ calculates for each interval-length Δt the possible number of events for the periodic event sequence Θ. An event stream is a specialized event sequence fulfilling the condition of sub-additivity. That means the shortest intervals have the highest densities of events. Formally, when an event bound function fulfills the condition of sub-additivity for every set of intervals $\Delta t_A, \Delta t_B$ the condition $\eta(\Delta t_A, \Theta) + \eta(\Delta t_B, \Theta) \geq \eta(\Delta t_A + \Delta t_B, \Theta)$ holds.

In chapter 4 this approach was extended to develop new fast sufficient and necessary schedulability analyses for systems with dynamic priority scheduling. The extension is based on the selectable approximation error. The approximation uses a large approximation error and achieves therefore a low run-time as long as possible and reduces the

error only when necessary to distinguish between schedulable and non-schedulable systems. The results are two new schedulability approaches, the dynamic-error algorithm and the all-approximation algorithm. The second one additionally returns to approximation as soon as possible after it was necessary to analyze one part exactly. Experiments with randomly generated task-sets show that these algorithms are much faster than the previous approaches (for some task sets by the order of magnitudes) and that the required run-time seems to be independent of the concrete values of the tasks. It seems only to depend on the number of tasks and their utilization.

In chapter 5 an approach for approximative and dynamic approximative analysis for systems with static priority scheduling were introduced. In contrary to previous solutions it is solely based on functions and allows an adaptive schedulability analysis. It required a new function; the exceeding cost function. It gives for each interval Δt that part of the requested costs within Δt that cannot be processed within Δt due to the later arrival time of its generating job. A job arriving two ms before the end of the interval and requiring five ms execution time will contribute at least 3 ms of them to the exceeding costs. Again the dynamic approximation outperforms the previous worst-case response time analysis also the results of the worst-case response time analysis might be more valuable as the new analysis achieves only a schedulable - non-schedulable decision. The exceeding cost function is a new idea that has shown to be also quite useful in later chapters.

In chapter 6 we have done experiments with randomly generated task sets to investigate the performance and the acceptance rates of the new algorithms and concepts. We have compared them with the performance of other existing approaches. The experiments give also some hints on the likely complexity of the algorithms.

In chapter 7 we have focused on the event model and to generalize the analysis approaches. The previous event stream model is quite powerful but it is not possible to describe bursts efficiently with this model. The event stream model requires a separate event element for each event of a burst. To overcome this disadvantage we have extended the event stream model to a hierarchical event spectra model.

We have introduced hierarchical event spectra consisting of hierarchical event elements. A hierarchical event element generates complete event patterns. These event patterns can be described either by an embedded (hierarchical) event spectrum or by a slope. Additionally, the hierarchical event elements have a limitation value L limiting the number of events that can be generated by the pattern within one period of the hierarchical event spectrum element. Important for an efficient calculation of the hierarchical event spectrum bound function is the separation condition, which prevents the overlapping of different periods of the same event element. We have provided the basic functions for the hierarchical event spectra. Note, that the previous event streams are only a special case of the hierarchical spectra. We have developed an approximation for the event spectra. The special characteristic of this approximation is that the number of test intervals for each embedded event spectra element is bounded globally. A hierarchical embedded event spectra element starts the generation of events from the beginning for each period of its parent spectra element. Bounding the number of test intervals for each spectrum element separately would

result in starting the exact evaluation again for each period of the parent. This would result in a global bound for maximum number of test intervals being exponential in the number of levels of the hierarchy.

We proposed a method that bounds the number of exactly evaluated test intervals globally for each event element. To do this we may require splitting the parent element near the maximum exact test interval of the child element. We have provided the method and the equations necessary to allow the global bound approximation.

The hierarchical event spectrum allows integrating the approximation in the spectra. Operations on these approximated spectra do not have to care of the approximation and the chosen degree of exactness but nevertheless benefit from the approximation. Last but not least the approximated hierarchical event spectra can be transformed into a curve consisting of simple consecutive piecewise linear curve segments in which the number of these elements is strictly limited by the approximation.

Concluding we have developed efficient schedulability analysis methods and an integrated schedulability analysis concept for uni-processor systems for both static and dynamic priority scheduling.

This work gives several opportunities for further research. First the integration of other scheduling algorithms into the framework is possible. This can be for example TDMA or Round-Robin. It would be necessary to extract how these algorithms modify the spectra. This would allow the analyzing of every possible combination of scheduling algorithms.

The event model and the algorithm can be used also for energy analysis [86] and energy saving [87] in embedded systems.

Another opportunity is the schedulability analysis for distributed systems. The different processing elements of such a system can have a multitude of scheduling algorithm to handle the tasks bound to them. They can be a mixture of scheduling with dynamic and static priorities. Task chains can be distributed on several elements requiring communication between the tasks and therefore between the processing elements.

The concept of event spectra can be extended to distributed systems. It is necessary to develop the transformation algorithm to extract outgoing event spectra of tasks from their incoming event spectra ([30]). Another interesting aspect for such an analysis is the dependencies between tasks on the same or different processing elements. Taking these dependencies into account can relax the schedulability problem and allow scheduling the tasks on less expensive processing elements.

In the real-time calculus it was proposed to model the flow between the tasks with arrival and service curves and equations were provided to calculate the outgoing curves of a task from the incoming ones. We used this concept in this work but we focus on the concrete possibility for an efficient implementation of this concept.

The proposed model and methodology offers many possibilities for further research. It may be interesting looking for them.

Zusammenfassung

Moderne Fahrzeuge haben oftmals mehr als 70 elektronische Steuergeräte (ECUs) welche miteinander kommunizieren müssen und eine Vielzahl unterschiedlicher Funktionen erfüllen. Diese Systeme müssen dabei nicht nur korrekte Ergebnisse liefern, sondern die Ermittlung der Ergebnisse muss meist innnerhalb fester Zeitschranken erfolgen, um rechtzeitige Reaktionen des Fahrzeuges auf eine sich veränderte Umwelt oder auf Steuerungsanweisungen des Fahrers sicherzustellen (Echtzeitanforderungen). Dabei sind in einer Funktionalität oftmals eine ganze Reihe von Steuergeräten involviert, welche über Busse miteinander kommunizieren. Aus Kostengründen ist es notwendig die Menge der Steuergeräte und Busse zu begrenzen, was dazu führt, dass sich viele Funktionen Steuergeräte und Busse teilen müssen und auf ihnen um die Rechenzeit und die Übertragungskapazität konkurrieren. Zur Steuerung dieser Konkurrenz werden verschiedene Ablaufplanungsverfahren (Schedulingstrategien) eingesetzt und den einzelnen Funktionen und Teilfunktionen (Tasks) teilweise unterschiedliche Prioritäten zugewiesen. Die sich daraus ergebenden möglichen zeitlichen Abläufe, Verdrängungen der verschiedenen Funktionen untereinander und letzlich der Antwortzeiten der einzelnen Funktionen und ihre Einhaltung von Echtzeiteigenschaften sind nicht einfach zu überblicken.

Verfahren der Echtzeitanalyse können für solche Systeme mit mathematischen Methoden die Einhaltung von Echtzeiteigenschaften verifizieren und obere und untere Schranken für Antwortzeiten bestimmen. Diese Verfahren ermöglichen somit fundierte Entscheidungen zur Dimensionierung der Steuergeräte, Verteilung der Funktionalität, Wahl der Ablaufplanungsverfahren und Verteilung der Prioritäten.

Ausgehend von einer zentralen Arbeit von Liu und Layland [88] wurden in den vergangenen 30 Jahren eine Reihe von Echtzeitanalyseverfahren und Echtzeitmodellen unterschiedlicher Qualität und Komplexität für verschiedene Ablaufplanungsverfahren sowohl für Ein-Prozessor-Systeme als auch für verteilte Systeme entwickelt.

Bei den Ablaufplanungsverfahren wird zwischen statischer Ablaufplanung, bei welchen die Tasks in einer vorab festgelegten Reihenfolge abgearbeitet werden und dynamischer Ablaufplanung, bei welcher die Anregungen der Tasks zusammen mit einer bestimmten Strategie den Ablaufplan der Tasks zur Laufzeit dynamisch bestimmt, unterschieden. Dynamische Ablaufplanungsverfahren kann man wiederum unterteilen in solche bei denen die Tasks eine feste Priorität bekommen und solche bei denen sich die Priorität dynamisch ändern kann. Als letztes wird unterschieden ob die Abarbeitung einer Task durch eine andere mit einer höheren Priorität unterbrochen werden kann (preemptives Scheduling) oder nicht (non-preemptives Scheduling).

In [88] wurde unter anderem gezeigt, dass das preemptive Ablaufplanungsverfahren Earliest-Deadline-First (EDF), ein Verfahren mit dynamisch veränderlichen Prioritäten, bei dem immer diejenige Task die höchste Priorität erhält, welche der absoluten Zeitschranke am nächsten liegt, ein optimales Ablaufplanungsverfahren darstellt, zumindestens sofern man den Aufwand für die Ablaufplanung selbst unberücksichtigt lässt. Optimalität bedeutet dabei, dass wenn es überhaupt einen Ablaufplan für das System gibt der alle Zeitanforderungen einhält, ein auf EDF basierender Ablaufplan auch alle Zeitanforderungen

einhalten würde. In der Praxis wichtiger ist, wegen ihrer einfacheren Implementierbarkeit, die Vergabe von festen Prioritäten als Ablaufplanung.

Bei den Echtzeitanalyseverfahren kann man zwischen exakten Verfahren, welche im Rahmen der Modellgenauigkeit exakt zwischen Systemen unterscheiden können, welche alle Echtzeitkriterien einhalten oder nicht, und hinreichenden Verfahren, welche nur die nicht-echtzeitfähigen Systemen korrekt klassifizieren können, unterscheiden. Vertreter für hinreichende Verfahren für Ein-Prozessor Systeme sind die Tests von Liu und Layland [88] für EDF und statische Prioritäten, der Test von Devi [46] für EDF und von Bini et al. [23] für statische Prioritäten. Für exakte Verfahren für Ein-Prozessor-Systeme seien die Antwortzeitanalyse für statische Prioritäten [73, 79] und das Processor-Demand-Criterion für EDF [19] genannt.

Analysen verteilter Systeme können entweder aus den exakten Analysen für Ein-Prozessor-Systemen aufgebaut werden, wie dies bei der Holistic-Scheduling Analyse und den darauf aufbauenden Verfahren der Fall ist oder einen in sich geschlossenen Ansatz wie beim Real-Time-Calculus verfolgen. Vom Real-Time-Caclulus gibt es wiederum eine hinreichende und eine exakte Variante.

Die Analyseverfahren benötigen jeweils Echtzeitmodelle welche eine Abstraktion der realen Systeme darstellen. Ein mögliches Modell ist die Darstellung des Systems über miteinander kommunizierende Tasks, wobei die Tasks durch Ereignisse angeregt werden. Den Tasks wird jeweils eine maximale Ausführungszeit (Worst-Case execution Time c^+) und eine minimale Ausführungszeit (Best-case Executione Time, c^-) zugeordnet. Zusätzlich können Tasks einen lokale Zeitschranke (deadline d) zugeordnet werden. Die Anregungen, welche von außen erfolgen können oder sich zwischen Tasks ergeben, können dabei durch Ereignismodelle beschrieben werden. Beispiele sind das Periodische Modell, das Periodische Modell mit Mindestabstand, das Ereignisstrommodell und die Real-Time Calculus Curves.

Im fortgeschrittenen periodischen Modell können die Anregungen durch eine Periode p und einen Jitter j beschrieben werden. Für jedes Ereignis gibt es ein Intervall der Länge j innerhalb dessen das Ereignis an einer beliebigen Stelle auftreten kann. Die Mittelpunkte der Intervalle folgen periodisch mit einem Abstand bzw. einer Periode p aufeinander. Vereinfacht gesprochen kommen die Ereignisse eigentlich periodisch mit Periode T können aber, im Rahmen ihres Jitterintervalls, etwas früher oder später kommen. Die Ereignisse haben im fortgeschrittenen periodischen Modell einen maximalen Abstand von $p+j$ und eine minimalen Abstand von $\max(0, p-j)$. Für ein Intervall der Länge Δt ergeben sich maximal $\left\lfloor \frac{\Delta t+j}{p} + 1 \right\rfloor$ und minimal $\max\left(0, \left\lfloor \frac{\Delta t-j}{p} + 1 \right\rfloor\right)$ Ereignisse.

Im periodischen Modell mit Mindestabstand kann zusätzlich noch ein Mindestabstand s zwischen zwei Ereignissen beschrieben werden. Dafür müssen die obigen Formeln nur geringfügig modifiziert werden. Unter der (sinnvollen) Bedingung $s \leq p$ ändern sich lediglich der minimale Abstand zwischen Ereignissen auf $\min(s, p-j)$ und die maximal sich für ein Intervall der Länge Δt ergebende Anzahl von Ereignissen auf:

$$\min\left(\left\lfloor \frac{\Delta t}{s} + 1 \right\rfloor, \left\lfloor \frac{\Delta t + j}{p} + 1 \right\rfloor\right)$$

Verwendet man nur das periodische Modell ohne Mindestabstand ergeben sich bei den Analysen keine falschen sondern nur zu konservative Ergebnisse, d.h. es werden weniger Systeme die alle Echtzeitbedingungen einhalten als solche erkannt.

Das Ereignisstrommodel wurde in [60, 61] von Gresser eingeführt. Es ermöglicht die genaue Modellierung komplexer Anregungen. Ein Ereignisstrom E besteht aus einer Menge von Tupeln $E = \{ES_1, ES_2, ..., ES_n\}$ wobei jedes Tupel durch eine Periode p und einen Offset a zu einem gemeinsamen Nullpunkt beschrieben wird $ES_i = \begin{pmatrix} p_i \\ a_i \end{pmatrix}$. Die Periode kann auch unendlich $(p_i = \infty)$ sein (aperiodische Elemente). Ein Sonderfall sind die homogenen Ereignisströme bei denen alle Elemente als Periode entweder eine gemeinsame Periode des Ereignisstroms oder unendlich haben. Die Anzahl der Ereignisse für ein Intervall der Länge Δt wird über die Ereignisstromfunktion $ES(\Delta t, E)$ berechnet, wobei $ES(\Delta t, E) = \sum_{ES_i \in E} ES(\Delta t, ES_i)$:

$$ES(\Delta t, ES_i) = \begin{cases} \left\lfloor \frac{\Delta t - a_i}{p_i} + 1 \right\rfloor & p_i \neq \infty \\ 1 & p_i = \infty \wedge \Delta t \geq a_i \\ 0 & p_i = \infty \wedge \Delta t < a_i \end{cases}$$

Die maximale und die minimale Menge von Ereignissen kann durch jeweils einen maximalen und einen minimalen Ereignisstrom beschrieben werden. Die Ereignisstromfunktion ermittelt für jede möglich Intervalllänge die Anzahl von Ereignissen. Die sich dabei ergebende monoton steigende Funtion ist essentiell für das Ereignisstrommodell und für die Echtzeitanalysetheorie. Die Modellerung der Ereignisströme muss so erfolgen, dass ein maximaler Ereignisstrom die Sub-Additivität erfüllt, d.h. $ES(\Delta t + \Delta t') \leq ES(\Delta t) + ES(\Delta t')$.

Im Real-Time Calculus werden die Ereignissmodelle durch maximale und minimale Arrival- und Service-Curves beschreiben. Diese Kurven sind Funktionen equivalent zur Ereignisstromfunktion. In der Theorie können diese Kurven jedes beliebige Ereignismodell exakt erfassen und beschreiben. Für die Anwendbarkeit werden konkrete Beschreibungsmodelle für diese Kurven benötigt. Es stehen bisher zwei konkrete Beschreibungsmodelle zur Verfügung. Im ersten Modell werden die Kurven jeweils durch maximal drei aufeinanderfolgende geradlinige Segmentelementen beschrieben. Dabei beschreibt das erste Segment den Bereich bis zum ersten Ereignis, das zweite Element einen initialen Burst und das dritte Element die langfristige Rate. Im zweiten exakten Beschreibungsmodell wird die Kurve in einen anfänglichen aperiodischen Teil und einen sich daran anschliessenden und sich wiederholenden periodischen Teil aufgeteilt. Jedes dieser Teile wird durch eine beliebig große Menge von aufeinanderfolgenden Segmentelementen beschrieben.

Das Ereignisstrommodell hat eine größere Modellierungsmächtigkeit und -genauigkeit als das fortgeschrittene periodische Modell. Die Genauigkeit einer Beschreibung mit Periode und Jitter lässt sich im Ereignisstrommodell mit vergleichbaren Aufwand erreichen. Der Mindestabstand überfordert allerdings die Modellierungsfähigkeiten des Ereignisstrommodells, wohingegen viele Anregungen mit dem Ereignisstrommodell genauer

erfasst werden können als mit dem periodischen Modell mit Mindestabstand. Das erste Beschreibungsmodell für den Real-Time Calculus hält die Berechnungskomplexität der Analyse niedrig, stellt allerdings eine sehr konservative Approximation des realen Systems dar. Das zweite Beschreibungsmodell kann beliebige reale Anregungen exakt abbilden, allerdings kann die Menge der zur Beschreibung benötigten Segmente schnell wachsen, so dass sich ein Komplexitätsproblem für die Echtzeitanalyse ergibt. Benötigt wird aber ein Ereignismodell welches gleichzeitig beschreibungsmächtig als auch effizient ist.

Die Echtzeitanalyseverfahren Antwortszeitanalyse und Processor Demand Criterion, welche ursprünglich beide für das fortgeschrittene periodische Modell entwickelt wurden, lassen sich einfach auf das periodische Modell mit Mindestabstand und das Ereignisstrommodell übertragen.

Das Processor Demand Criterion zur Analyse von Ein-Prozessor-Systemen mit EDF-Scheduling funktioniert wie folgt. Es wird für einen konkreten Ablaufplan untersucht ob alle Zeitschranken eingehalten werden. Start dieses Ablaufplans ist eine synchrone Aktivierung aller Tasks, d.h. alle Tasks werden unabhängig voneinander zeitgleich aktiviert. Sind die Tasks unabhängig, kann eine solche Situation tatsächlich auftreten, ansonsten stellt sie eine obere Schranke für die dichtestmögliche Aktivierung dar. Dieser Ablaufplan wird nun mathematisch simuliert und jede Zeitschranke wird auf ihre Einhaltung überprüft. Diese Simulation und Überprüfungen geschehen in einem Intervall, das von der synchronen Aktivierung einerseits und einer Testgrenze andererseits begrenzt wird. In der Literatur wurden verschiedene Testgrenzen entwickelt und jeweils nachgewiesen, dass für nicht-echtzeitfähige Systeme Echtzeitverletzungen innerhalb des durch die Testgrenze begrenzten Intervalls auftreten müssen. Beispiele sind die in [17] vorgestellte Testgrenze

$$\Delta t_{max} = \max_{\forall \tau \in \Gamma}(p_\tau - d_\tau)\frac{U_\Gamma}{1-U_\Gamma}$$

mit U_Γ als Auslastung des Systems und die [119] in vorgestellte Testgrenze:

$$\Delta t_{max} = \frac{\sum_{\forall \tau \in \Gamma}\left(1-\frac{d_\tau}{p_\tau}\right)c_\tau^+}{1-U_\Gamma}$$

Allen Testgrenzen gemeinsam ist die Abhängigkeit ihrer Lage von den konkreten Parametern der Ereignismodelle. So ist die erste Testgrenze vom maximalen Abstand $p_\tau - d_\tau$ abhängig, die zweite vom maximalen Verhältnis von der Zeitschranke d_τ zur Periode p_τ abhängig. Die Anzahl der in einem Intervall zu untersuchenden Zeitschranken hängt für jede Task vom Verhältnis des durch die Testgrenze begrenzten Intervalls zur Periode p_τ der Task ab. Hat man in einem System Tasks mit sehr kurzen und sehr langen Perioden oder Zeitschranken, so kann dies zu einer späten Testgrenze, somit einem langen Testintervall mit gleichzeitig sehr vielen zu untersuchenden Zeitschranken führen. Daraus ergibt sich ein hoher Rechenaufwand für die Analyse. Dieses Problem ergibt sich aber für alle drei Ereignismodelle gleichermaßen. Dies führt selbst für Systeme mit einer begrenzten maximalen Auslastung zu einer pseudo-polynomialen Analysekomplexität. Der Analyseaufwand ist somit nicht nur von der Anzahl der Tasks im System und der Auslastung sonder zusätzlich von den konkreten Parametern der Task und dem zugehörigen

Ereignismodell abhängig. Dies macht die Laufzeit für allgemein verwendbare Werkzeuge zur Echtzeitanalyse nicht voraussehbar und kann zu hohem Rechenaufwand führen.

Ziel dieser Arbeit ist die beiden vorgenannten Probleme grundsätzlich zu lösen.

In dieser Arbeit machen wir dafür die folgenden Beiträge:

- Wir stellen das erste approximative Echtzeitanalyseverfahren mit polynomialer Komplexität vor, dessen maximaler Analyseaufwand unabhängig ist von den Parametern des Echtzeit- und Ereignismodells, sondern nur von der Anzahl der Tasks und der gewählten Genauigkeit abhängt und dessen Genauigkeit sich nur durch die Kapazität der gewählten Prozessoren ausdrücken lässt.
- Wir stellen ein neues Ereignismodell, die hierarchischen Ereignisspektren vor, welches die Modellierungsmächtigkeit der oben genannten Ereignismodelle vereinigt und denoch zu einer kompakten und effizienten Beschreibung führt.
- Wir wenden die Approximation auf das neue Ereignismodell an, wobei die Approximation in die Beschreibung des Ereignismodells vollständig integriert wird. Das resultierende Modell kann zu einer einfachen aber, dank der Approximation, dennoch kompakten neuen Beschreibung für die Kurven des Real-Time Calculus ausgerollt werden.

Daneben werden noch die folgenden Beiträge gemacht:

- Ein effizientes approximatives Echtzeitanalyseverfahren für EDF.
- Ein auf die Dynamisierung der Approximation beruhendes Analyseverfahren für EDF welches teils um Größenordnungen schneller arbeitet als alle bisher bekannten Analyseverfahren.
- Ein neues effizientes Analyseverfahren für Prioritätsscheduling für Ein-Prozessor-Systeme.
- Mehr Verständnis für die Zusammenhänge zwischen den einzelnen Modellen und für die Theorie der Echtzeitanalyse.

Im folgenden stellen wir den ersten Beitrag, die approximative Echtzeitanalyse am Beispiel eines mit EDF geplanten Ein-Prozessor-Systems vor. In der Approximation wird, durch gezielte Einführung von Ungenauigkeit in die Analyse, der Aufwand für dieselbe reduziert. In Abbildung 10.0.1 ist die Idee der Approximation dargestellt. Zu sehen ist die Ereignisfunktion für eine Task und die vorgeschlagene Approximation derselben. Ebenfalls dargestellt ist die verfügbare Kapazität in Form der Winkelhalbierenden. Die Ereignisfunktion ist für das periodische Modell eine einfache Treppenfunktion wobei die Treppenbreite durch die Periode der Task und die Treppenhöhe durch die maximale Ausführungszeit gegeben ist. Um die Echtzeiteigenschaften zu beweisen wird die Rechenzeitanforderungsfunktion mit der verfügbaren Kapazität verglichen (vgl. Processor-Demand-Criterion). Dazu muss die Rechenzeitanforderungsfunktion an allen für den Vergleich relevanten Werten (Testpunkten) berechnet werden. Relevant sind dabei alle Treppenstufen. Es müssen alle Testpunkte bis zum maximalen Testintervall als obere Schranke berücksichtigt werden. Der Aufwand der Analyse hängt von der Anzahl der Testpunkte, also der

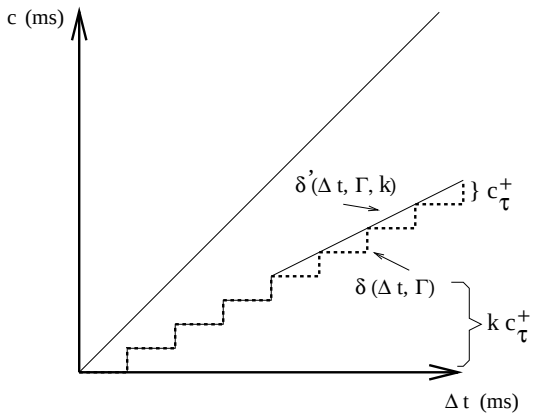

FIGURE 10.0.1. Approximation einer einfachen Task

Anzahl der Treppenstufen ab. Die approximierte Ereignisfunktion ist für die ersten k Treppenstufen identisch mit der exakten Ereignisfunktion. Die weiteren Treppenstufen werden durch eine Gerade approximiert. Die Steigung der Geraden entspricht dabei der relativen Auslastung der Resources durch diese Task ($\frac{c^+}{p}$).

Die Gerade berührt die exakte Ereignisfunktion bei jeder Treppenstufe. Die daraus resultierende maximale Abweichung der approximierten Rechenzeitanforderungsfunktion von der exakten Rechenzeitanforderungsfunktion beträgt einmal c^+. An dem Punkt, an dem die approximierte Funktion von der exakten Funktion beginnt abzuweichen, beträgt der Funktionswert mindestens kc^+ da für die ersten k Treppenstufen beide Funktionen identisch sind. Daraus ergibt sich ein maximaler relativer Fehler von:

$$\varepsilon = \frac{c^+}{kc^+} = \frac{1}{k}$$

Der Fehler ist nur abhängig vom gewählten Approximationsfaktor k und nicht von den Parametern der Task c^+ und p. Interessant wird die Approximation erst bei mehreren Tasks. Dabei werden mathematisch die approximierten Rechenzeitanforderungsfunktionen der einzelnen Tasks addiert. Algorithmisch kann die Berechnung der addierten Funktionen gemeinsam erfolgen. Entscheidend ist, dass sowohl die Anzahl der Testpunkte als auch die Approximationsgenauigkeit bei der Addition der approximierten Funktionen der einzelnen Tasks erhalten bleibt. Für Tasks mit kleiner Periode (und daher meist kleiner maximaler Ausführungszeit) beginnt die Approximation früher, führt aber auch nur zu einer kleinen absoluten Ungenauigkeit, bei Tasks mit großen Periode (und dabei meist langen maximalen Ausführungszeiten) beginnt die Approximation erst später, wenn die größere absolute Ungenauigkeit sich relativiert. Somit wird bei einer gewählten Genauigkeit von k die Anzahl der maximal zu berechnenden Stützpunkte der Funktion auf nk begrenzt wobei n die Anzahl der Tasks ist. Die maximale Ungenauigkeit bleibt bei $\frac{1}{k}$.

Die neue Approximation fügt sich nahtlos in die existierenden Echtzeitanalyseverfahren für EDF ein. Es kann bewiesen werden, dass das Verfahren von Devi einen Spezialfall der Approximation mit einem Approximationsfaktor von $k = 1$ darstellt. Das Processor-Demand-Criterion ist natürlich auch nur ein Spezialfall der Approximation mit $k = \infty$.

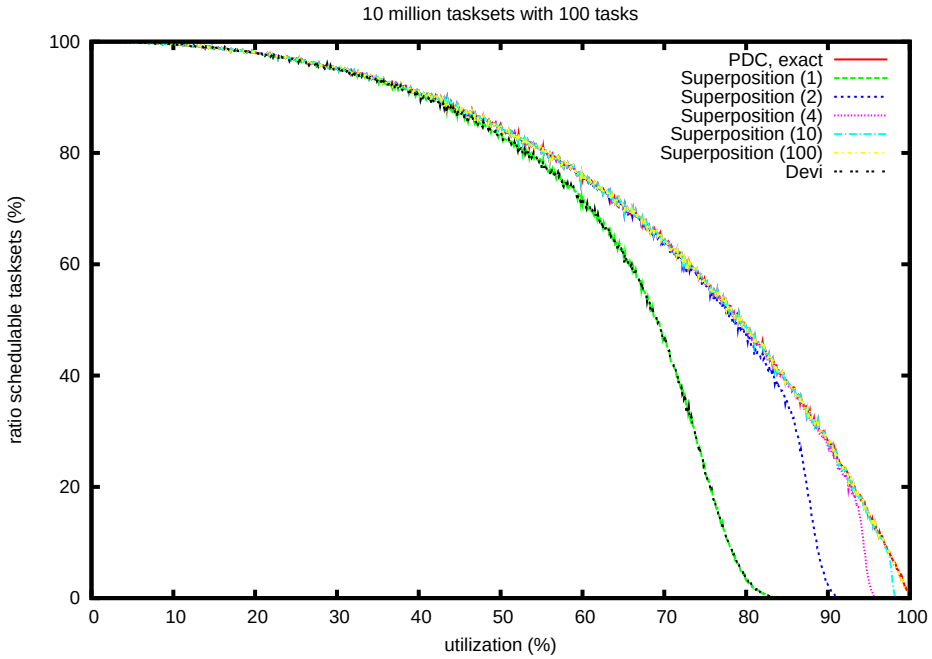

FIGURE 10.0.2. Anteil der als planbar klassifizierten Tasksets

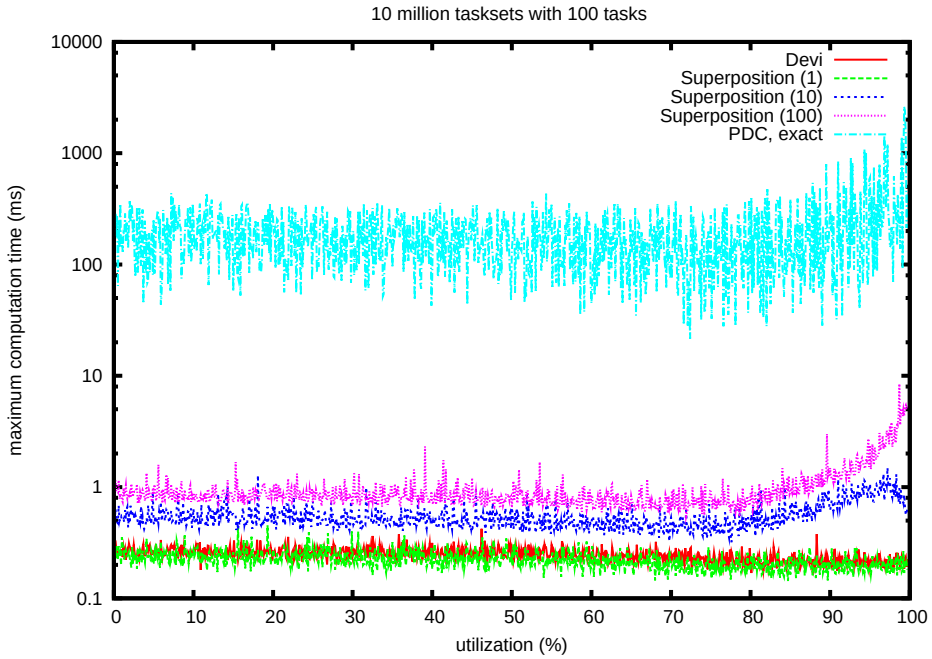

FIGURE 10.0.3. Maximale Laufzeit der Echtzeitanalysen für verschieden Approximationstufen

In Abbildung 10.0.2 wird die Genauigkeit und in Abbildung 10.0.3 die Laufzeit der

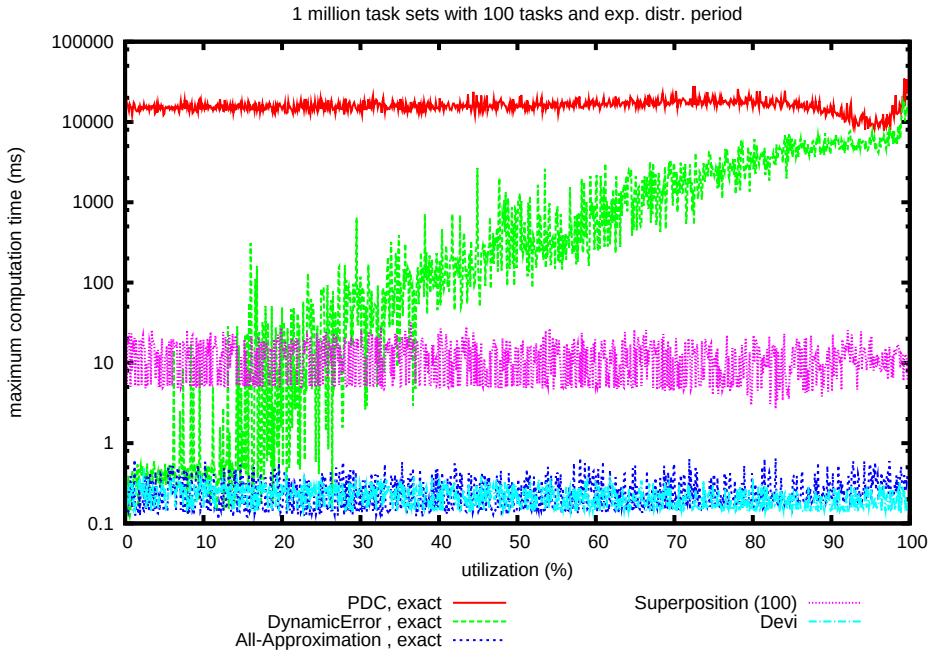

FIGURE 10.0.4. Rechenzeiten der dynamischen Approximation und des Processor-Demand-Criterion

approximativen Analyse für verschiedene Werte von k dargestellt und mit anderen Echtzeitanalyseverfahren verglichen. Für die Genauigkeit wird dabei der Anteil der von dem jeweiligem Verfahren als echtzeitfähig klassifizierten Systemen im Verhältnis zu den gesamten untersuchten Systemen dargestellt. Das Processor-Demand-Criterion begrenzt dabei, als exaktes Verfahren, den Raum der echtzeitfähigen Systeme. Auffällig ist, dass schon für relative kleine Werte von k nahezu alle echtzeitfähig Systeme auch als solche klassifiziert werden. Die Rechenzeit der Approximation ist hingegen geringer als die der exakten Analyse.

Durch eine Dynamisierung der Approximation ergeben sich neue schnelle exakte Echtzeitanalyseverfahren. Bei der Dynamic-Error Analyse wird die Analyse zunächst mit relativ geringer Genauigkeit ($k = 1$) gestartet und nur wenn notwendig wird die Genauigkeit schrittweise gesteigert. Dadurch können Systeme mit hohen Echtzeitreserven schnell analysiert werden und nur für Systeme im Grenzbereich zur Nichtechtzeitfähigkeit werden längere Analysezeiten benötigt. Die All-Approximation Analyse geht noch einen Schritt weiter und wendet die Approximation immer an soweit sie möglich ist.

In Abbildung 10.0.4 wird die Rechenzeit dieser neuen exakten Analyse mit der Rechenzeit des Processor-Demand-Criterion verglichen. Es zeigt sich, dass die Rechenzeiten insbesondere der All-Approximation Analyse die Rechenzeit der bisher besten Analyse teilweise um Größenordnungen unterschreitet.

Die Approximation dient ebenfalls als Grundlage für neue Echtzeitanalyseverfahren für Ein-Prozessor-Systeme mit statischen Prioritäten.

Um die Probleme des Ereignisstrommodells zu überwinden wird ein neues Ereignismodell, die hierarchischen Ereignisspektren, vorgeschlagen. Es ermöglicht eine effiziente Modellierung von allen Arten von Ereignisschüben (Bursts). Ein hierarchisches Ereignisspektrum $\hat{\Theta}$ besteht aus einer Menge von hierarchischen Ereignisspektrumelementen $\hat{\theta}$ mit $\hat{\theta} = (p, a, L, f, \hat{\Theta}')$ wobei p die Periode, a den Offset, L die Begrenzung der in einer Periode maximal von dem Ereignisspektrumelement erzeugbaren Kosten, f eine Erzeugungsrate von Kosten und $\hat{\Theta}'$ ein Sub-Ereignisspektrum darstellt. L begrenzt dabei die entweder durch f oder durch $\hat{\Theta}'$ in einer Periode erzeugbaren Kosten, wobei per Definition in einem Element nur entweder die Erzeugungsrate oder das Sub-Element aktiv ist (also entweder $f = 0$ oder $\hat{\Theta}' = \emptyset$). Ebenfalls per Definition wird die Seperationsbedingung gefordert, welche einer Überlappung verschiedener Perioden eines Ereignisspektrumelements verhindert. Dabei wird die Modellierungsmächtigkeit der hierarchischen Ereignisspektren nicht eingeschränkt aber eine effiziente Berechnung der zugehörigen Funktionen ermöglicht.

Die hierarchischen Ereignisspektren ermöglichen auch eine Integration der Approximation in das Echtzeitmodell. Damit können bei der Propagierung von Ereignisspektren diese gezielt durch Approximation vereinfacht werden und nur die vereinfachten Spektren in der weiteren Analyse verarbeitet werden, ohne dass in der weiteren Analyse die verwendete Approximationgenauigkeit bekannt sein muss.

Bibliography

[1] K. Albers. Erweiterung eines multikriteriellen optimierungsverfahrens für eingebettete systeme um ein verfahren zur echtzeitanalyse. Master's thesis, Friedrich-Alexander-Universität Erlangen, 2002.

[2] K. Albers, F. Bodmann, and F. Slomka. Hierachical event streams and event dependency graphs. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 97–106, 2006.

[3] K. Albers, F. Bodmann, and F. Slomka. Run-time efficient feasibility analysis of uni-processor systems with static priorities. In *Poceedings of the International Embedded Systems Symposium (IESS 2007)*, 2007.

[4] K. Albers, F. Bodmann, and F. Slomka. Advanced hierarchical event-stream model. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS'08)*, 2008.

[5] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *IEEE Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*, pages 187–195, Catania, 2004.

[6] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with edf-scheduling. In *Proceedings of the Design Automation and Test Conference in Europa (DATE'05)*, pages 492–497, 2005.

[7] J. Anderson, P. Holmann, and A. Srinivasan. *Handbook of Scheduling*, chapter Fair Scheduling of Real-Time Tasks on Multiprocessors. Chapman & Hall, 2004.

[8] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. In *Software Engineering Journal*, 1993.

[9] N.C. Audsley, A.R. Burns, M.F. Richardson, and A.J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*. IEEE Computer Society Press, 1991.

[10] J. Axelsson. *Analysis and Synthesis of Heterogeneous Real-Time Systems*. Phd thesis, Linköping, 1997.

[11] S. Baruah and N. Fisher. The feasibility analysis of multiprocessor real-time systems. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 85–94, 2006.

[12] S.K. Baruah. A general model for recurring real-time tasks. In *Proceedings of the Real-Time Systems Symposium*, pages 114–122, Madrid, 1998.

[13] S.K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *International Journal of Real-Time Systems*, 24:98–128, 2003.

[14] S.K. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *The International Journal of Time-Critical Computing Systems*, 17:5–22, 1999.

[15] S.K. Baruah and N. Fisher. The partitioned scheduling of the sporadic real-time tasks on multiprocessor platforms. In *Proceedings of the International Conference on Parallel Processing Workshop (ICPPW'05)*, 2005.

[16] S.K. Baruah and J. Goossens. *Handbook of Scheduling*, chapter Scheduling Real-Time Tasks: Algorithm and Complexity. Chapman & Hall, 2004.

[17] S.K. Baruah, R.R. Howell, and L.E. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, sporadic, real-time tasks on one processor. *International Journal of Real-Time Systems*, 2:301–324, 1990.

[18] S.K. Baruah, R.R. Howell, and L.E. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118:3–20, 1993.

[19] S.K. Baruah, A. Mok, and L.E. Rosier. Preemptive scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the Real-Time Systems Symposium*, pages 182–190, 1990.

[20] I. Bates and A. Burns. An integrated approach to scheduling in safty-critical embedded control systems. In *Real-Time Systems*, volume 25, pages 5–37, 2003.

[21] G. Bernat. Response time analysis of asynchronous real-time systems. In *Real-Time Systems*, volume 25, pages 131–156, 2003.

[22] E. Bini and S. Baruah. Efficient computation of response time bounds under fixed-priority scheduling. In *Proceedings of the 15th International Conference on Real-Time and Network Systems*, 2007.

[23] E. Bini and G.C. Buttazzo. The space of rate monotonic schedulability. In *Proceedings of the 23th Real-Time Systems Symposium*, 2002.

[24] E. Bini and G.C. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*. IEEE Computer Society Press, 2004.

[25] E. Bini and G.C. Buttazzo. Scheduling analysis for periodic fixed priority systems. In *IEEE Transactions on Computers*, number 53(11), pages 1462–1473, 2004.

[26] E. Bini and G.C. Buttazzo. Measuring the performance of schedulability tests. In *Real-Time Systems*, volume 30 (1-2), pages 129–154. IEEE Computer Society Press, May 2005.

[27] E. Bini, G.C. Buttazzo, and G.M. Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*. IEEE Computer Society Press, 2001.

[28] E. Bini, Giorgio. C. Buttazzo, and Giuseppe Buttazzo. Rate monotonic analysis: the hyperbolic bound. In *IEEE Transactions on Computer*, number 52(7), pages 933–942, 2003.

[29] T. Blickle, J. Teich, and L. Thiele. Systems-level synthesis using evolutionary algorithms. *Design Automation For Embedded Systems*, 3(1):23–58, 1998.

[30] F. Bodmann, K. Albers, and F. Slomka. Analyzing the timing characteristic of task activations. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems (IES'06)*, 2006.

[31] J-Y. Le Boudec and P. Thiran. Network calculus: A theory of determinstic queuing systems for the internet. In *Lecture Notes in Computer Science*. Springer, 2001.

[32] R. Bril, W. Verhaegh, and E. Pol. Initial values for on-line response time calculation. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 13–22, 2003.

[33] A.R. Burns and A.J. Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley, 2nd edition, 1996.

[34] G.C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic, 1997.

[35] Giorgio C. Buttazzo. Rate monotonic vs. edf: judgment day. In *Real-Time Systems*, volume 29, pages 5–26, Norwell, MA, USA, 2005. Kluwer Academic Publishers.

[36] J. Carpenter, S. Funk, P. Holmann, A. Srinivasan, J. Anderson, and S.K. Baruah. *Handbook of Scheduling*, chapter A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithm. Chapman & Hall, 2004.

[37] S. Chakraborty, T. Erlebach, and L. Thiele. On the complextiy of scheduling conditional real-time code. TIK Report 107, ETH Zürich, 2001.

[38] S. Chakraborty, S. Künzli, and L. Thiele. Approximate schedulability analysis. In *23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pages 159–168, 2002.

[39] S. Chakraborty, S. Künzli, and L. Thiele. Performance evaluation of network processor architectures: Combining simulation with analytical estimations. *Computer Networks*, 41(5):641–665, 2003.

[40] S. Chakraborty and L. Thiele. A new task model for streaming applications and its schedulability analysis. In *IEEE Proceedings of the Design Autionation and Test in Europe Conference (DATE'05)*, pages 486–491, 2005.

[41] D. Chen, A. K. Mok, and S. Baruah. On modeling real-time task systems. In *Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems*, pages 153–169, 1996.

[42] D. Chen, A. K. Mok, and T.-W. Kuo. Utilization bound revisited. In *Proceedings of the 6th Real-Time Computing Systems and Applications*, pages 295–302, 1999.

[43] R.L. Cruz. A calculus for network delay. In *IEEE Transactions on Information Theory*, volume 37, pages 114–141, 1991.

[44] B.P. Dave and N.K. Jha. COHRA: Hardware-software co-synthesis on hierarchical heterogeneous distributed embedded systems: Hardware-software cosynthesis on hierarchical heterogeneous distributed embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):900–919, 1998.

[45] B.P. Dave, G. Lakshminarayana, and N.K. Jha. COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI)*, 7(1):92–104, 1999.

[46] U. Devi. An improved schedulability test for uniprocessor periodic task systems. In *IEEE Proceedings of the 15th Euromicro Conference on Real-Time Systems*. IEEE

Computer Society Press, 2003.

[47] R. Devillers and J. Goossens. General response time computation for the deadline driven scheduling of periodic tasks. In *Fundamenta Informaticae*, volume 40, pages 199–219, 1999.

[48] R.P. Dick and N.K. Jha. MOGAC: A multiobjective genetic algorithm for hardware-software co-synthesis of distributed embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):920–935, 1998.

[49] M. Dörfel, A. Mitschele-Thiel, and F. Slomka. CORSAIR: HW/SW-codesign von kommunikationssystemen mit SDL. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 23(1):3–13, 2000.

[50] R. Ernst, M. Jersak, K. Richter, and F. Slomka. Transformation of sdl specifications for systems-level timing analysis. In *International Symposium on Hardware/Software Co-Design*, Estes Park, 2002.

[51] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasilbility analysis in static-priority systems. In *Work-in-Progress Proceedings of the IEEE International Real-Time Systems Symposium*, Lissabon, December 2004.

[52] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, July 2005.

[53] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. In *Proceedings of the 13th International Conference on Real-Time Systems*, Paris, April 2005.

[54] N. Fisher, S. Baruah, and T. P. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 118–127, 2006.

[55] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uni-processor scheduling. Rapport de Recherche RR-2966, INRIA, 1996.

[56] S. Goddard. *On the Management on Latency in the Synthesis of Real-Time Processing Systems from Process Graphs*. Ph. d. dissertation, University of North Carolina at Chapel Hill, Department of Computer Science, 1998.

[57] S. Goddard and X. Liu. A variable rate execution model. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 135–143, 2004.

[58] J. Goossens. Worst case response time versus worst case offset configuration using the deadline driven scheduler. In *Proceedings of the 9th International Conference on Real-Time Systems*, pages 123–132, 2001.

[59] J. Goossens. Scheduling of offset-free systems. In *Real-Time Systems*, volume 24, pages 239–258, 2003.

[60] K. Gresser. *Echtzeitnachweis ereignisgesteuerter Realzeitsysteme*. Phd thesis (in german), Düsseldorf, 1993.

[61] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, 1993.

[62] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with symta/s-symbolic timing analysis for systems. In *Proceedings of 25th International Real-Time Systems Symposium (RTSS'04)*, December 2004.

[63] C.-C. Han and H.-Y. Tyan. A better polynomial-time schedulability test for real-time fixed priority scheduling algorithms. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 36–45, 1997.

[64] W. Henderson, D. Kendell, and A. Robson. Improving the accuracy of scheduling analysis applied to distributed systems. *International Journal of Time-Critical Computing Sysemts*, 20:5–25, 2001.

[65] R. Henia and R. Ernst. Context-aware scheduling analysis of distributed systems with tree-shaped task-dependencies. In *Proceedings of the Design Automation and Test Conference in Europa (DATE'05)*, pages 480–485, 2005.

[66] J. Hromkovic. *Algorithmics for Hard Problems*. Texts in Theoretical Computer Science. Springer, 2nd edition, 2003.

[67] K. Jeffay and S. Goddard. A theory of rate-based execution. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 304–314, Phoenix, 1999.

[68] D. Jelkmann, K. Albers, and F. Slomka. Improved feasibility tests for asynchronous real-time periodic task sets. In Christian Haubelt and Jürgen Teich, editors, *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 2007.

[69] M. Jersak. *Compositional Performance Analysis for Complex Embedded Applications*. Phd thesis, TU Braunschweig, 2005.

[70] M. Jersak and R. Ernst. Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. In *Proceedings 40th Design Automation Conference (DAC'03)*, Juny 2003.

[71] M. Jersak, R. Henia, and R. Ernst. Context-aware performance analysis for efficient embedded system design. In *Proceedings Design Automation and Test in Europe (DATE'04)*. IEEE Computer Society Press, 2004.

[72] M. Jersak, K. Richter, and R. Ernst. Performance analysis for complex embedded applications. *International Journal of Embedded Systems*, 2004. Special Issue on Codesign for SoC.

[73] M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Computer Journal*, (29 (5)):390–395, 1986.

[74] S. Kollmann, K. Albers, F. Bodmann, and F. Slomka. Modifications of event streams for the real-time analysis of distributed fixed-priority systems. In *13th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'06)*, Potsdam, 2006.

[75] S. Kollmann, K. Albers, and F. Slomka. Dependencies aware event-driven real-time analysis for distributed fixed-priority systems. Internal Report 289-vts-60593, Ulm University, 2007.

[76] S. Künzli. *Efficient Design Space Exploration for Embedded Systems*. Phd thesis, ETH Zürich No. 16589, 2006.

[77] S. Künzli, A. Hamann, R. Ernst, and L. Thiele. Combined approach to system level performance analysis of embedded systems. In *Poceedings of the 5th IEEE/ACM International Confernence on Hardware/Software Codesign and System Synthesis*, pages 63–68, 2007.

[78] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1991.

[79] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS'90)*, pages 201–209, 1990.

[80] J. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behaviour. In *Proceedings of the Real-Time Systems Symposium*, 1989.

[81] J. Lehoczky, L. Sha, J. Strosnider, and H. Tokuda. Fixed priority scheduling theory for hard real-time systems. In *Foundations of Real-Time Computing: Scheduling and Resource Management*, pages 1–30. Kluwer Academic Publishers, 1991.

[82] J. Leung, editor. *Handbook of Scheduling: Algorithm, Models and Performance Analysis*. Chapman & Hall, 2004.

[83] J. Leung and M.Merril. A note on preemptive scheduling of periodic real-time tasks. In *Information Processing Letter*, volume 11, pages 115–118, 1980.

[84] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[85] G. Lipari and G.C. Buttazzo. Schedulability analysis of periodic and aperiodic tasks with ressource constraints. *Journal of System Architecture*, (46):327–338, 2000.

[86] H. Lipskoch, K. Albers, and F. Slomka. Battery discharge aware energy feasibility analysis. In *Proceedings of the 4th international conference on Hardware/ Software Codesign and system synthesis*, pages 22–27, New York, NY, USA, November 2006. ACM Press.

[87] H. Lipskoch, K. Albers, and F. Slomka. Fast calculation of permissable slowdown factors for hard real-time systems. In *Proceedings of the 17th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'07)*, number 4644 in Springer Lecture Notes in Computer Science (LNCS), 2007.

[88] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time enviroments. *Journal of the ACM*, 20(1):46–61, 1973.

[89] H. Liu and X. Hu. Efficient performance estimation for general real-time task systems. In *International Conference on Computer Aided Design*, pages 464–471, 2001.

[90] J.W.S. Liu. *Real-Time Systems*. Prentice-Hall Inc., Upper Saddle River, New Jersey, 2000.

[91] J. Maki-Turja and M. Nolin. Efficient response-time analysis for tasks with offsets. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium*

(RTAS'04), pages 462–471, 2004.

[92] Y. Manabe and S. Aoyagi. A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling. *Real-Time Systems*, (14):171–181, 1998.

[93] A. Masrur, S. Drössel, and G. Färber. Improvements in polynomial-time feasibility testing for edf. In *Proceedings of the Design Automation and Test Conference in Europa (DATE'08)*, pages 1033–1038, 2008.

[94] A. Masrur and G. Färber. Ideas to improve the performance in feasibility testing for edf. In Tei-Wei Kuo, editor, *Proceedings Work-In-Progress Session of the 18th Euromicro Conference on Real-Time Systems*, pages 17–20, 2006.

[95] A. Maxiaguine, S. Künzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *Proceeding Design Automation and Test in Europa (DATE'04)*, pages 1040–1045. IEEE Computer Society, 2004.

[96] A. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10), 1997.

[97] M. Di Natale and J. Stankovic. Dynamic end-to-end guarantees in distributed real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 216–227, 1994.

[98] J. C. Palencia and M. González Harbour. Offset-based response time analysis of distributed systems scheduled under edf. In *IEEE Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, 2003.

[99] J. C. Palencia Gutiérrez, J. J. Gutiérrez García, and M. González Harbour. On the schedulability analysis for distributed hard real-time systems. In *IEEE Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, pages 136–143, 1997.

[100] J. C. Palencia Gutiérrez, J. J. Gutiérrez García, and M. González Harbour. Best case analysis for improving the worst-case schedulability test for distributed hard real-time systems. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, pages 35–44, Berlin, 1998.

[101] A.K. Parekh and R.G.Gallager. A generalized processor sharing approach to flow control in integrated service networks. In *IEEE/ACM Transactions on Networking*, volume 1, pages 344–357, 1993.

[102] D. Park, S. Natarajan, and M.J. Kim. A generalized utilization bound test for fixed-priority real-time scheduling. In *Proceedings of the 2nd International Workshop on Real-time Systems and Applications*, pages 73–76, 1995.

[103] M. Park and Y. Cho. An efficient feasibility test method for hard real-time periodic tasks. In *The 21th IEEE Real-Time Systems Symposium*, 2000.

[104] R. Pellizzoni and G. Lipari. A new sufficient feasibility test for asynchronous periodic real-time task sets. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 204–211, 2004.

[105] R. Pellizzoni and G. Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Journal of Real-Time Systems*, 30(1-2):105–108, 2005.

[106] R. Pellizzoni and G. Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. In *11th IEEE Real-Time and Embedded*

Technology and Application Systems, pages 65–75, 2005.

[107] P. Pop, P. Eles, and Z. Peng. Schedulability analysis for heterogenous time/event triggered real-time systems. In *IEEE Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, 2003.

[108] P. Pop, P. Eles, Z. Peng, and V. Izosimov. Schedulabilityy-driven partitioning and mapping for multi-cluster real-time systems. In *IEEE Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*, 2004.

[109] P. Pop, P. Eles, Z. Peng, and T. Pop. Scheduling and mapping in an incremental design methodology for distributed real-time systems. In *IEEE Transactions on Very Large Scale Integration (VLSI)*, volume 12(8), pages 793–811, 2004.

[110] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *11th IEEE Real-Time and Embedded Technology and Application Symposium*, pages 160–169, 2005.

[111] O. Redell. Accounting for precedence constraints in the analysis of tree-shaped transcations in distributed real-time systems. trita-mmk 4, Royal Institute of Technology (KTH), Stockholm, 2003.

[112] O. Redell. *Response Time Analysis for Implementation of Distributed Control Systems*. Phd thesis, trita-mmk 17, Royal Institute of Technology (KTH), Stockholm, 2003.

[113] O. Redell. Analysis of tree-shaped transactions in distributed real time systems. In *IEEE Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04))*, pages 239–248, 2004.

[114] O. Redell and M. Sanfridson. Exact best-case response time analysis of fixed prority scheduled tasks. In *Proceedings of the 14th International Conerence on Real-Time Systems (ECRTS'02)*, pages 165–172, Vienna, 2002.

[115] O. Redell and M. Törngren. Calculating exact worst-case response times for static priority scheduled tasks with offsets and jitter. In *Proceedings of the 8th Real-Time and Embedded Technology and Application Symposium (RTAS'02)*, pages 164–172, San Jose, 2002.

[116] P. Richard. On the complexity of scheduling real-time tasks with self-suspension on one processor. In *IEEE Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, 2003.

[117] K. Richter. *Compositional Scheduling Analysis Using Standart Event Models*. Phd thesis, TU Braunschweig, 2005.

[118] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *Proceedings of the Design Automation and Test Conference in Europe (DATE'02)*, 2002.

[119] I. Ripoll, A. Crespo, and A. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Systems*, 11(1):19–39, 1996.

[120] M. Rudorfer, T. Ochs, M. Thiede, M. Missmer, O. Scheickl, and H. Heinecke. Realtime system design using autosar methodology. In *Elektronik automotive: Special issue AUTOSAR*, 2007.

[121] P. Scholz. *Softwareentwicklung eingebetteter Systeme*. Springer, 2005.

[122] M. Sjödin and H. Hansson. Improved response-time analysis calculations. In *IEEE Proceedings of the 19th Real-Time Systems Symposium (RTSS'98)*, 1998.

[123] F. Slomka. New techniques for the design of distributed embedded real-time systems. In *Proceedings of the Embedded World Conference*, 2005.

[124] F. Slomka. Simulation of distributed embedded real-time systems. In *13th GI/ITG Conference on Measurement, Modeling and Evaluation of Computer and Communication Systems*, pages 449–452, 2006.

[125] M. Spuri. Analysis of deadline scheduled real-time systems. Interner Bericht 2772, INRIA, 1996.

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

[127] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design space exploration for the network processor architectures. In *1st Workshop on Network Processors at the 8th International Symposium for High Performance Computer Architectures*, 2002.

[128] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the IEEE Conference of Circiuts and Systems*, 2000.

[129] K. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS 221, University of York, Dep. of Computer Science, England, 1994.

[130] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Mircoprogramming*, 40(23):117–134, 1994.

[131] E. Wandler. *Modular performance analysis and interface-based design for embedded real-time systems*. Phd-thesis nr. 16819, ETH Zürich, 2006.

[132] E. Wandler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event steams in analysis of hard real-time applications. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS'04)*, pages 450–459, Toronto, 2004.

[133] E. Wandler and L. Thiele. Characterizing workload correlations in multi processor hard real-time systems. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS'05)*, pages 46–55, San Francisco, 2005.