Real-Time Schedulability Analysis with Formal Techniques

Youcheng Sun

Scuola Superiore Sant'Anna, Pisa

Dissertation submitted to Scuola Superiore Sant'Anna for the degree of Doctor of Philosophy.

Supervisor:

Giuseppe Lipari
University of Lille

Committees:

Enrico Bini
Scuola Superiore Sant'Anna

Giorgio Buttazzo
Scuola Superiore Sant'Anna

Marko Bertogna
University of Modena

Marco Di Natale
Scuola Superiore Sant'Anna

Gilles Geeraerts
Université Libre de Bruxelles

Contents

List of Figures

Chapter 1

Introduction

Scheduling is the art of distributing the capacity provided by a resource to competing entities. In particular, *real-time scheduling* deals with deterministically distributing time to (real-time) tasks subject to temporal constraints. In the context of real-time scheduling, the time typically refers to the processing time in modern computing systems.

The simplest, yet expressive, characterisation for a real-time task is of the form $\tau_i = (C_i, D_i, T_i)$, where C_i is the task's Worst-Case Execution Time (WCET), D_i is the relative deadline, and T_i regulates τ_i's activations. A task can release an infinite sequence of task instances called *jobs*. $J_{i,k}$ denotes the k-th job of τ_i, which is further characterised by the release time (or arrival time) $r_{i,k}$ and the absolute deadline $d_{i,k} = r_{i,k} + D_i$ such that a job $J_{i,k}$ must execute up to C_i time units no later than $d_{i,k}$. For a *periodic* task τ_i, T_i is its period and for any k, $r_{i,k+1} - r_{i,k} = T_i$; for a *sporadic* task τ_i, T_i measures the minimum time distance between two successive jobs of τ_i: for any k, $r_{i,k+1} - r_{i,k} \geq T_i$. The finishing time of $J_{i,k}$ is denoted by $f_{i,k}$ and $f_{i,k} - r_{i,k}$ is a job's response time. Then, the Worst-Case Response Time (WCRT) of a task τ_i can be defined as $R_i = \max_{\forall k}\{f_{i,k} - r_{i,k}\}$. A task is said to be schedulable if for its every job there is $f_{i,k} \leq d_{i,k}$, that is, $R_i \leq D_i$.

The scheduler is another entity in charge of deciding which tasks to execute at each time instant. A *real-time system* refers to the tasks, the single processor or multiprocessor platform and the scheduler. The fundamental problem in a real-time system is to verify the *schedulability* of each task under the scheduling algorithm, and this is called *real-time schedulability analysis*.

Assume that there are two periodic tasks $\tau_1 = (2, 4, 4)$ and $\tau_2 = (5, 10, 10)$ executing on a single processor. Roughly speaking, τ_1 requests 2 time units every 4 time units, and τ_2 requests 5 time units every 10 time units. According to the Rate Monotonic (RM) scheduling policy [LL73], a higher priority is assigned to the faster task that is with shorter period. As a result, τ_1 enjoys a higher priority than τ_2, and the arrival of τ_1 can preempt the execution of τ_2. The snapshot of one possible schedule of the two tasks by RM scheduling policy, in the time interval $[0, 10)$, is shown in Figure 1.1.

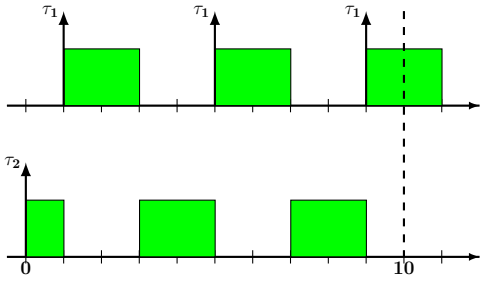


Figure 1.1: A possible schedule of two tasks in the time interval $[0, 10)$

The history of real-time scheduling can be traced back to the work of Liu and Layland in 1973 [LL73], which is the first important paper in real-time scheduling theory. In [LL73], the Fixed Priority (FP) scheduling and the Earliest Deadline First (EDF) scheduling of real-time tasks upon a single processor platform are proposed. After more than 40 years, the FP policy and the EDF policy are still today's most popular real-time scheduling algorithms.

In the FP scheduling, each task is statically assigned a fixed priority in advance and the arrival of a higher priority task can preempt the execution of a lower priority task. In the EDF scheduling, the priority of each job is dynamically adjusted given its absolute deadline and an earlier deadline corresponds to a higher priority.

More importantly, Liu and Layland contribute the fundamental insight for single processor real-time schedulability analysis, which is known as the *Critical Instant Theorem*. A critical instant for a task is the release time that results in the worst-case scenario for its execution. Under the FP scheduling, the critical instant is the time point at which all tasks are simultaneously activated. For the simple two-task FP system discussed above, the worst-case scenario for τ_2's execution is shown in Figure 1.2, and τ_2 is not schedulable.

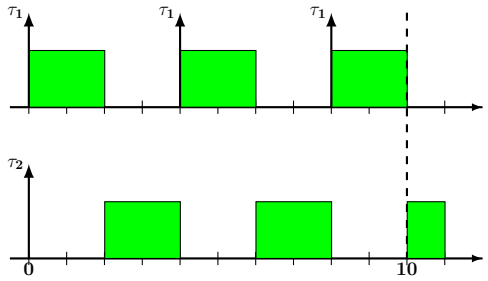


Figure 1.2: The worst-case scenario in the single processor FP scheduling

As a matter of fact, the critical instant depicts a special moment when scheduling a set of real-time tasks, and it is enough to analyse the schedulability from this particular moment concerning temporal constraints. By focusing on the worst-case scenario resulted from the critical instant and instead of considering each possible task activation pattern, the schedulability analysis is then greatly simplified. Additionally, even if the critical instant never happens, analyses can consider the pessimistic assumption of the critical instant as a sufficient condition for the schedulability. In fact, many existing tests are only sufficient, for reducing the complexity.

Liu and Layland's observation of the critical instant and the worst-case scenario is simple but fundamental. Since then, a large body of research literature has addressed the problem of schedulability analysis of real-time tasks, by looking for the worst-case scenario in a real-time system. However, the de facto truth about real-time schedulability analysis is that there is only one task activation pattern that is well studied and recognised: *all tasks are simultaneously activated, and the subsequent task instances are released As Soon As Possible (ASAP)!*

Due to successful application of the ASAP task activation pattern in a variety of real-time systems, the majority of schedulability tests make the effort to derive "efficient enough" mathematical formulas from this pattern for checking the schedulability. As a result, these formulas fall into certain forms. In the following, this family of schedulability tests is regarded as the *analytical approach* for schedulability analysis, and the analytical schedulability analysis represents the main results in the literature of real-time scheduling theory in the past 40 years.

Details on the analytical schedulability analysis can be found in textbooks like [Liu00] and [But11]. As an example, the Response Time Analysis (RTA) [ABR+93] is one fundamental methodology for schedulability analysis and it has been applied to every well-known real-time system. The RTA employs

an iterative procedure to compute a task's worst-case response time so as to decide whether the task is schedulable or not. Its basic form is as follows.

$$X = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{X}{T_j} \right\rceil C_j \tag{1.1}$$

Indeed, Equation 1.1 corresponds to the RTA procedure for a task τ_i subject to single processor FP scheduling, where a lower task index denotes a higher priority. It is derived by analysing the ASAP task release pattern starting from the critical instant. The computational complexity for this RTA test is pseudo polynomial. It is widely accepted that a schedulability test is "efficient enough" if it has the complexity that is not worse than pseudo polynomial time.

On the other side, there exist real-time systems such that they do not have the critical instant or worst-case scenario for the exact/sufficient schedulability analysis. Along with the trend of switching from the single processor architecture to the multiprocessor architecture, more and more real-time scheduling problems cannot be naively explained by a worst-case scenario like the ASAP pattern. For example, upon the multiprocessor platform, a task may be allowed to migrate among different processors. This is called the global scheduling of real-time tasks, and the schedulability analysis of multiprocessor global scheduling is still an open problem.

When in a real-time system the worst-case scenario does not exist, or is too pessimistic, or is too difficult to find, the classic analytical approach is not the niche for schedulability analysis any more. Nowadays, finding new solution techniques becomes the major challenge for advancing the theory of real-time schedulability analysis. This thesis's main effort is to integrate the analytical approach and the use of formal methods for real-time schedulability analysis.

Besides the analytical approach, real-time schedulability analysis can be also done by using formal methods, based on Timed Automata [AD94] or Linear Hybrid Automata [ACHH93]. The schedulability problem is usually encoded as a reachability problem on the state space of the formal model: *bad states* that represent a violation of the temporal guarantee should never be reached. Generally speaking, the reachability analysis in many formal models is not decidable, that is, the analysis procedure cannot be guaranteed to terminate. However, as shown in this thesis, most schedulability problems are decidable when formally modeling them.

In contrast to the analytical approach, the formal approach for schedulability analysis is able to detect each possible state when scheduling a set of real-time tasks, and can be applied to general systems with or without the worst-case scenario. Moreover, for real-time systems that cannot be simply represented by a worst-case scenario, the formal approach provides an alternative to help understand what indeed happens in the system.

However, to explore all states in a formal model can be expensive regarding time complexity and memory cost, and the state space explosion problem is so notorious that the adoption of formal methods for real-time schedulability analysis is resistant. The state space explosion is the fact that the number of states in a model can be so large and it is beyond the power of modern computing devices to conduct the reachability analysis (even in case the procedure is decidable). For example, the reachable states in a Linear Hybrid Automata model are typically computed using polyhedra [HPR97], but the size of polyhedra representations can be exponential in the number of variables in the model.

To mitigate the state space explosion problem, domain knowledge and experience, largely resulted from the analytical reasoning, on real-time scheduling can be utilised. Every state in the state space encodes a particular moment for the scheduling of tasks, and it is the responsibility of algorithm designers to investigate if there are some states that are more critical than others regarding the violation of temporal constraints or to cut off states that are not useful for schedulability analysis. In the extreme case, the

so-called worst-case scenario results in a series of system states that are enough for deciding a task's schedulability. By continually complicating a real-time system (e.g. increasing the number of tasks or processors), the formal schedulability test will eventually fail to apply. Nevertheless, it is meaningful to study to which extent an integration of the analytical approach and the formal approach works.

After Liu and Layland's initial work and due to the somewhat heuristic worst-case scenario appearing in many single processor real-time systems, the analytical approach has been dominating the real-time schedulability analysis and the way people interpret a real-time scheduling problem. The message conveyed in this thesis is that advanced results by different solution techniques should be integrated for better real-time schedulability analysis and better understanding of real-time scheduling problems.

Organisation of the thesis At first, Chapter 2 introduces the common notations and concepts that will be used in the context of this thesis. In Chapter 3, a formal framework for modeling the single processor schedulability analysis is proposed. Then, the parametric schedulability analysis and the generation of timed interfaces for compositional system design are discussed. Chapter 4 focuses on component-based schedulability analysis in hierarchical systems such that the schedulability test of tasks within a component does not rely on other parts of the system. After Chapter 4, the platform under schedulability analysis moves from the single processor to the multiprocessor; more specifically, the multiprocessor global scheduling will be considered. In Chapter 5, an exact global FP schedulability analysis based on an automaton model is proposed. On one side, empirical knowledge from real-time scheduling helps reduce the complexity of the exact analysis; on the other side, inherent features of global FP scheduling are obtained when applying formal analysis on the model. Chapter 6 recalls general knowledge and state-of-the-art results regarding global schedulability analysis based on the analytical approach. Then, Chapter 7 and Chapter 8 present new analytical techniques for global FP scheduling and global EDF scheduling respectively. Chapter 9 presents the challenges for schedulability analysis in discrete time domain and a methodology is developed to tackle them. Finally, the thesis is concluded in Chapter 10.

This thesis is based on the following articles.

1. Youcheng Sun, Giuseppe Lipari, Étienne André, and Laurent Fribourg. **Toward Parametric Timed Interfaces for Real-Time Components.** Proceedings of the 1st International Workshop on Synthesis of Continuous Parameters (SynCoP), 2014.

2. Youcheng Sun, Giuseppe Lipari, Romain Soulat, Laurent Fribourg, and Nicolas Markey. **Component-Based Analysis of Hierarchical Scheduling using Linear Hybrid Automata.** Proceedings of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014.

3. Youcheng Sun and Giuseppe Lipari. **A Pre-Order Relation for Exact Schedulability Test of Sporadic Tasks on Multiprocessor Global Fixed-Priority Scheduling.** Real-Time Systems.

4. Youcheng Sun, Giuseppe Lipari, Nan Guan, and Wang Yi. **Improving the Response Time Analysis of Global Fixed-Priority Multiprocessor Scheduling.** Proceedings of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014.

5. Youcheng Sun and Giuseppe Lipari. **Response Time Analysis with Limited Carry-in for Global Earliest Deadline First Scheduling.** Proceedings of the 36th IEEE Real-Time Systems Symposium (RTSS), 2015.

6. Youcheng Sun and Giuseppe Lipari. **The Task Parameter Scalability Problem in Real-Time Systems.**

Chapter 2

Preliminary Definitions

This chapter presents the common concepts and notations that will appear in the rest of the thesis.

2.1 System Model

A real-time task is denoted by τ_i and is characterised by the tuple (C_i, D_i, T_i).

- C_i is the Worst-Case Execution Time.
- D_i is the relative deadline of τ_i.
- In case τ_i is a *periodic* task, T_i represents the period of the task such that two successive task activations are strictly separated by T_i; or, if τ_i is a *sporadic* task, T_i measures the minimum time distance between two successive activations of τ_i. With an abuse of notation, we also call T_i the period even for a sporadic task τ_i.

A task τ_i is said to have an *implicit deadline* if $D_i = T_i$; A *constrained-deadline* task τ_i refers that $D_i \leq T_i$; similarly, an *unconstrained-deadline* task means $D_i > T_i$; in the end, if we say a task has an arbitrary deadline, then its relative D_i may be less than, equal to, or larger than T_i.

A task τ_i periodically or sporadically activates. Each task activation is denoted by a job $J_{i,k}$, where the index k refers to the k-th instance of τ_i. Each job $J_{i,k}$ is further characterised by its release time (or arrival time) $r_{i,k}$ and absolute deadline $d_{i,k}$ such that $d_{i,k} = r_{i,k} + D_i$. The finishing time $f_{i,k}$ of a job is the time $J_{i,k}$ finishes the requested computation and the difference between $f_{i,k}$ and $r_{i,k}$ is called the response time of $J_{i,k}$. The Worst-Case Response Time (WCRT) R_i of a task τ_i is the maximum response time among all its jobs such that $R_i = \max_{\forall k}\{f_{i,k} - r_{i,k}\}$. A task is *schedulable* if every of its jobs finishes execution no later than the corresponding absolute deadline, that is, $R_i \leq D_i$. A job is said to be *active* if it has been released by it has not finished the execution. For an unconstrained-deadline task, there may be more than one active job at the same time and it is required that all jobs from the same task must execute sequentially, that is, a job can start executing only after all its precedent jobs from the same task finish the execution.

The utilisation of a task τ_i is defined $U_i = \frac{C_i}{T_i}$ and for simplicity, it is required that $C_i \leq \min\{D_i, T_i\}$.

A task set $\mathcal{T} = \{\tau_i, \ldots, \tau_n\}$ contains n real-time tasks. The total utilisation of the task set is denoted as $U_{tot} = \sum_{1 \leq i \leq n} U_i$. The hyperperiod of a task set is the *least common multiple* of all task periods: $H = \mathsf{lcm}(T_1, \ldots, T_n)$. By default, a task is not allowed to self-suspend its execution, and tasks are independent from each other such that they do not share mutual exclusion resources.

This thesis mainly focuses on two kinds of scheduling policies: the Fixed Priority (FP) scheduling and the Earliest Deadline First (EDF) scheduling. Under the FP policy, each task is assigned a fixed

priority in prior and all its jobs inherit this priority. By convention, a lower task index refers to a higher priority; that is, τ_i has a higher priority than τ_j if $i < j$. As for the EDF policy, the priority of each job is decided by its absolute deadline and a shorter deadline corresponds to a higher priority. In both cases, the task preemption is allowed. A preemption means that an executing task gives up the processor because that a higher priority job is released.

When it comes to the multiprocessor platform, we assume there are m ($\leq n$) identical processors. The *global* scheduling policy features the facts that a job may execute upon any processor and a preempted job may later resume execution upon the same processor as, or upon a different processor from, the one it had been executing. Thus, the FP and EDF scheduling policies are then adapted as Global FP (G-FP) and Global EDF (G-EDF).

Other task models A task τ_i with explicit initial offset is denoted as $\tau_i = (O_i, C_i, D_i, T_i)$ such that O_i specifies the release time of the first job from τ_i, and C_i, D_i, and T_i keep their original meanings. Still, τ_i can be a periodic task or sporadic task.

The *arrival curve* [TCN00] is a generalisation of the sporadic arrival model. In this case the pattern of arrival must respect a certain function called *arrival curve* $\alpha_i(t) : \mathbb{R} \to \mathbb{N}$. The arrival curve constrains the number $\mathfrak{N}$ of task arrivals in any interval of a given length Δ:

$$\forall k \geq 0, \forall \mathfrak{N} > 0 : \Delta \leq \alpha_i(r_{i,k+\mathfrak{N}-1} - r_{i,k})$$

In other words, the number of jobs from a task τ_i in any interval must not exceed the value of the arrival curve for that interval[1]. In particular, a periodic arrival curve is of the form:

$$\alpha_{N^u,P}(t) = N^u + \left\lfloor \frac{t}{P} \right\rfloor \tag{2.1}$$

where N^u denotes the initial burstiness and P denotes the period. A generic arrival curve can always be upper bounded by a periodic arrival curve of the form (2.1).

2.2 Linear Hybrid Automata

Throughout the thesis, when it refers to formal techniques, in principle it means modeling the scheduling problem or schedulability analysis in hybrid automata and then applying modeling checking techniques for verifying certain property related with the task system's schedulability. A hybrid automaton [ACHH93, HPR97] is a finite automaton associated with a finite set of variables continuously varying in dense time. In this section, the basic terminology and the definition of Linear Hybrid Automata will be introduced.

Let $\mathsf{Var} = \{x_1, \ldots, x_n\}$ be a set of *continuous variables* (sometimes called *clocks*) and $\dot{\mathsf{Var}} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ be the set of variables' *derivatives* (also called *rates*) over time. A *linear constraint atom* over Var is of the form $\sum_{i=1}^{n} c_i x_i \sim b$, where c_i ($1 \leq i \leq n$) and b are rational numbers and $\sim \in \{<, \leq, =, \geq, >\}$. A *linear constraint* $\mathcal{C}$ is the conjunction of a finite number constraint atoms. A *valuation* ν over Var is a function that assigns a real value to each element in Var. The set of all possible valuations over Var is denoted as $V(\mathsf{Var})$. We write $\nu \models \mathcal{C}$ to represent that ν satisfies $\mathcal{C}$. The same notations can also be defined for $\dot{\mathsf{Var}}$.

Definition 1. *A Linear Hybrid Automaton (LHA)* $\mathcal{A} = (\mathsf{Var}, \mathsf{Loc}, \mathsf{Init}, \mathsf{Lab}, \mathsf{Trans}, D, \mathsf{Inv})$ *consists of seven components:*

[1] In the original paper [TCN00], a lower bound on the number of arrival events is also considered.

- *a finite set* Var *of continuous variables;*
- *a finite set* Loc *of locations including an initial location* l_0*;*
- *a labeling function* Init *that specifies the initial linear constraint over continuous variables;*
- *a finite set* Lab *of synchronisation labels including a* stutter *label* ϵ*;*
- *a finite set* Trans *of transitions;*
- *a labeling function* D *that assigns to each location* $l \in$ Loc *a linear constraint over variables' derivatives;*
- *and a labeling function* Inv *that assigns each location* $l \in$ Loc *a linear constraint, called* invariant*, over variables.*

The automaton can be in a location l as long as the current valuations of the variables satisfy $\mathsf{Inv}(l)$. A *transition* (also called *edge*) is a tuple $(l, \gamma, a, \alpha, l')$ consisting of a source location l, a target location l', a guard γ that is a linear constraint over Var, a synchronisation label $a \in$ Lab, and the transition relation α which is used to update the values of the variables in Var. By default, it is required that on each location, there is a *stutter transition* $(l, true, \epsilon, Id, l)$, where $Id = \{(\nu, \nu) | \nu \in V(\mathsf{Var})\}$ is the identical transition relation.

Figure 2.1 depicts a LHA model for a water tank monitor ([HPR97]): $\mathsf{Var} = \{\mathrm{w}, \mathrm{t}\}$, $\mathsf{Loc} = \{$pump_is_on, stop_pump, pump_is_off, start_pump, error$\}$ with $l_0 =$ pump_is_on, $\mathsf{Init}(\text{pump\_is\_on}) = \mathrm{t} = 0 \wedge \mathrm{h} \leq \mathrm{w} \leq \mathrm{H}$, and $\mathsf{Lab} = \{\mathrm{stop}, \mathrm{off}, \mathrm{start}, \mathrm{on}, \mathrm{overflow}, \mathrm{underflow}\}$; as an example for the invariant, there is $\mathsf{Inv}(\text{pump\_is\_on}) = \mathrm{w} \leq \mathrm{H}$.

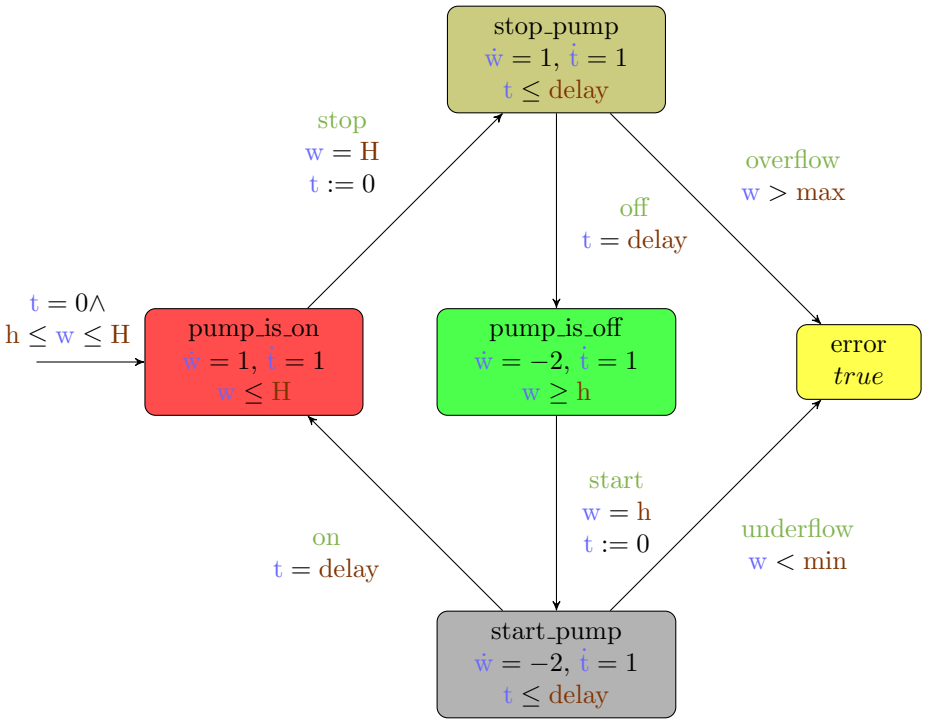


Figure 2.1: The LHA model for a water tank: $\mathrm{h} = 5, \mathrm{H} = 10, \mathrm{delay} = 2, \mathrm{min} = 1$ and $\mathrm{max} = 12$

Committed locations A committed location is a special location within which the time elapsing is not allowed. Figure 2.2 shows the graphical representation (with double circles) of a committed location with name $\mathrm{loc_x}$.

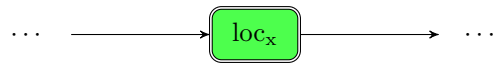


Figure 2.2: A committed location

For convenience, we allow to put two synchronisation labels on the same transition. This means that the two synchronisation actions must be triggered without time elapsing, which can be implemented by inserting a committed location in between.

LHA is very expressive and in fact most real-time scheduling problems can be modeled by a subclass of LHA. The Timed Automaton [AD94] is a special kind of LHA such that continuous variables always have derivatives equivalent to 1 and if a variable is updated along with a transition, it can only be reset to 0. The Timed Automaton with Stopwatches (or Stopwatch Timed Automaton) [CL00] extends the Timed Automaton such that a continuous variable can be *stopped* in some locations, that is, the corresponding variable rate equals to 1.

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two LHA over a set of variables Var. Their parallel composition $\mathcal{A}_1 \times \mathcal{A}_2$ is the LHA $(\mathsf{Var}, \mathsf{Loc}_1 \times \mathsf{Loc}_2, \mathsf{Init}, \mathsf{Lab}_1 \cup \mathsf{Lab}_2, \mathsf{Trans}, D, \mathsf{Inv})$ as follows.

- $\mathsf{Init}(l_1, l_2) = \mathsf{Init}(l_1) \wedge \mathsf{Init}(l_2)$.
- $((l_1, l_2), \gamma, a, \alpha, (l_1', l_2')) \in \mathsf{Trans}$ iff
  1. $(l_1, \gamma_1, a_1, \alpha_1, l_1') \in \mathsf{Trans}_1$ and $(l_2, \gamma_2, a_2, \alpha_2, l_2') \in \mathsf{Trans}_2$;
  2. $\gamma = \gamma_1 \wedge \gamma_2$;
  3. either $a_1 = a_2 = a$, or either $a_1 = a \notin (\mathsf{Lab}_1 \cup \mathsf{Lab}_2)$ and $a_2 = \epsilon$ or $a_1 = \epsilon$ and $a_2 = a \notin (\mathsf{Lab}_1 \cup \mathsf{Lab}_2)$;
  4. $\alpha = \alpha_1 \wedge \alpha_2$.
- $D(l_1, l_2) = D_1(l_1) \wedge D_2(l_2)$;
- $\mathsf{Inv}(l_1, l_2) = \mathsf{Inv}_1(l_1) \wedge \mathsf{Inv}_2(l_2)$.

Semantics in the LHA A *concrete state* s of the LHA is of the form (l, ν), where l is a location and $\nu \in V(\mathsf{Var})$. A state can change by the following two ways.

- A discrete step: $(l, \nu) \xrightarrow{a} (l', \nu')$ which means there exists a transition $(l, \gamma, a, \alpha, l')$ and
$$\nu \models \gamma \wedge \nu' = \alpha(v) \wedge \nu' \models \mathsf{Inv}(l').$$
- A time step: $(l, \nu) \xrightarrow{t} (l, \nu')$, where t is a real value represents the time elapsed.
$$\nu \models \mathsf{Inv}(l) \wedge \nu' \in \nu \uparrow^t_{D(l)} \wedge \nu' \models \mathsf{Inv}(l) \wedge t \geq 0.$$
$\nu \uparrow^t_{D(l)}$ represents the set of valuations that can be reached by letting variables continuously evolve for t time units, according to derivatives constrained by $D(l)$, and starting from the valuation ν. When $t = \infty$, for simplicity, we use $\nu \nearrow D(l)$ to denote $\nu \uparrow^t_{D(l)}$.

A *generic step* is denoted as $\rightarrow$ and it can be either a discrete step or time step. A sequence of steps is represented as $\Rightarrow$, and $\overset{t}{\Rightarrow}$ means that the accumulated time during the sequence of steps is t.

A *symbolic state* S of the LHA is a pair $(l, \mathcal{C})$, where l is a location and $\mathcal{C}$ is a linear constraint over variables in Var. Geometrically, a linear constraint can be mapped to a convex region; the notation $\mathcal{C}$ is also used to denote this convex region. The concepts of a step and a sequence of steps can also be defined

for the symbolic state by lifting the definitions for concrete states. When it comes to symbolic states, the corresponding operations are performed on convex regions instead of concrete valuations. Given a sequence of symbolic states, by abstracting away the constraints on variables, the result is an alternating sequence of locations and synchronisation labels and this is called a *trace*.

Algorithm 1: Reachability Analysis in a LHA

```
1:  R ← {S_0}
2:  while true do
3:    P ← Post(R)
4:    if P ∩ F ≠ ∅ then
5:      return No
6:    end if
7:    R' ← R ∪ P
8:    R' ← Max^⊇(R')
9:    if R' = R then
10:     return Yes
11:   else
12:     R ← R'
13:   end if
14: end while
```

The initial state S_0 of a LHA can be defined as $(l_0, \mathcal{C}_0)$ with

$$\mathcal{C}_0 = \mathsf{Init}(l_0) \nearrow D(l_0) \ \wedge \mathsf{Inv}(l_0).$$

Given an arbitrary state $S = (l, \mathcal{C})$, we say it is reachable in LHA if there exists a sequence of steps from S_0 to S and $\mathcal{C}$ is not empty, and the corresponding trace is said to be admissible.

One fundamental problem in LHA is to check if certain locations or states, e.g., representing the deadline miss conditions, are reachable. The basic procedure of reachability analysis is as in Algorithm 1.

R denotes the set of reachable states in SA and F is the set of states that should be not reached. The Post operation returns the set of states that can be reached in a single transition from states in R. If some state in F is reachable, then the whole procedure terminates.

$\mathsf{Max}^{\supseteq}(R')$ is defined as

$$\forall S \in \mathsf{Max}^{\supseteq}(R') : \nexists S' \in R' \ s.t. \ S'.l = S.l \wedge S'.\mathcal{C} \supset S.\mathcal{C}$$
$$\forall S, S' \in \mathsf{Max}^{\supseteq}(R') : \neg(S'.l = S.l \wedge S'.\mathcal{C} \supseteq S.\mathcal{C})$$

In fact, for two states S and S' such that $S.l = S'.l \wedge S.\mathcal{C} \supseteq S'.\mathcal{C}$, the reachable states from S' would be a subset of the reachable states from S.

In general, the reachability analysis in LHA is not decidable: if it terminates, then it returns correct results, but there is no guarantee that the analysis procedure will terminate. Nevertheless, as we are going to see in the following content of the thesis, the schedulability analyses in LHA are usually decidable.

LHA with parameters The parametric LHA extends LHA with a finite set U of parameters. A *parameter* has a constant value but the exact value is unknown. Or, a parameter can be regarded as a special kind of continuous variable that is always with derivative 0. In this sense, a parametric LHA is in fact also a LHA. However, it is meaningful to distinguish parametric LHA from plain LHA as it specialises in certain problems. For example, if in the LHA model as in Figure 2.1 values of min and max become unknown, then people may be interested in asking if there exist parameter valuations of min

and max such that the location error is not reachable, or finding all possible parameter valuations that keep the error not reachable. Given a parametric LHA $\mathcal{A}$ and a parameter valuation π, $\mathcal{A}[\pi]$ denotes the LHA by instantiating parameters in $\mathcal{A}$ with π.

Chapter 3

Parametric Schedulability Analysis

In this chapter, a framework in Stopwatch Timed Automata to model single processor Fixed Priority (FP) scheduling of real-time tasks will be presented. The framework is modular, so as to be easily adapted to different schedulers and task models. By parameterising the model, the parametric schedulability analysis can be conducted. We first show that such an analysis does not provide satisfactory results when task periods are considered as unknown parameters. After identifying and investigating the problem, a solution is proposed by adapting the model to take into account the worst-case scenario in schedulability analysis. The parametric analysis in the modified model can always converge when the task set total utilisation is strictly less than 100%. Then, we show how to use our parametric analysis for the generation of timed interfaces in compositional system design. Finally, we discuss how to more efficiently perform parametric analysis.

The content in this chapter is based on [SLAF14].

3.1 Introduction

It is possible to address the problem of schedulability analysis of real-time tasks by both using analytical approach (e.g. [Liu00, But11]) and formal methods (e.g. [AM02, FPY02, FMPY03]). However, in these works, the task parameters (e.g. the execution time, relative deadline or period) are required to have explicit values. By admitting some task parameters to be unknown, parametric schedulability analysis synthesises parameter valuations such that the task set is guaranteed to be schedulable. This permits to explore the design space parameters, and to assess the robustness of the system.

Analytical parametric analysis of real-time systems using mathematical equations has already been done in the past. Bini et al. [BDNB08] proposed a method for parametric analysis of real-time periodic tasks where unknown parameters can be either WCETs or task periods. [SSL+14] extended the parametric analysis in [BDNB08] to tasks with arbitrary deadlines and the distributed real-time system. However, with such approaches, changing the task model requires the development of a new methodology. Also, such methods do not apply to the case of arbitrary unknown parameters; for example, a task's WCET and period can be unknown parameters at the same time.

It is possible to perform an exploration of the parameter space using Timed Automata, as in [LPPR13]. However, their approach is not fully parametric: the analysis is repeated for all combinations of the discrete values of the parameters.

Fully parametric analysis can be performed using specific formalisms. For example, formalisms such as parametric Timed Automata [AHV93] and parametric PEtri nets [TLR09] have been used to model parametric schedulability problem (e.g. [CPR08,SSL+14]). Particularly, thanks to the generality of these modeling languages, it is possible to perform fully parametric analysis on arbitrary task parameters.

The Inverse Method (IM) [AS13] can be used for exploring the space of parameters of a parametric Stopwatch Timed Automata in the proximity of a valuation point of parameters. In this chapter, we will utilise this method for the parametric schedulability analysis.

A similar approach has been used in [SSL+14], where a distributed real-time system has been modeled using parametric Stopwatch Timed Automata. However, in [SSL+14] the methodology is limited to consider tasks' computation times as unknown parameters. Here, tasks' periods and relative deadlines can also be unknown.

Moreover, our generic modular approach can be seen as contract-based methodology where "provided" and "required" interfaces are instances of *(assumption, guarantee)* pairs in the contract terminology. An interface-based approach to the design and analysis of real-time systems using assume/guarantee has already been proposed in the literature [LPT13,SL08], but their approach is not parametric. Compositional verification of timed systems, using assume/guarantee reasoning, has also been considered in [LAL+14] for Event-Recording Automata, a subclass of Timed Automata; again, this approach is non-parametric.

3.2 The Inverse Method

The Inverse Method (IM) for parametric Stopwatch Timed Automata exploits the knowledge of a specific valuation of parameters, called the *reference point*, for which the behaviour of the system is known. The method synthesises automatically a dense space of valuations around the reference point, for which the discrete behaviour of the system, that is the set of all the admissible traces, is guaranteed to be the same.

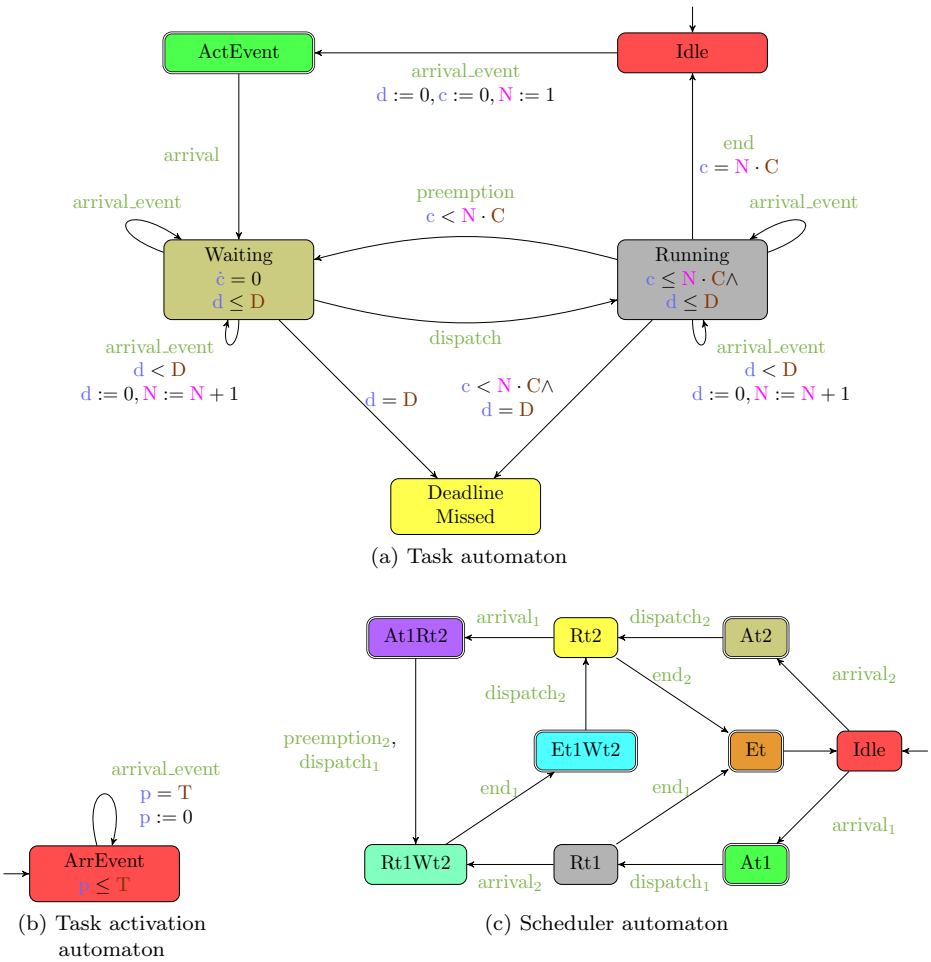
Given a reference point π, the IM proceeds by exploring iteratively the state space from the initial state. When a π-incompatible state is met (that is a state $(l, \mathcal{C})$ such that $\pi \not\models \mathcal{C}$), a π-incompatibility inequality J is selected within the projection of $\mathcal{C}$ to parameters U. The inequality is then negated, and the analysis restarts with a model further constrained by $\neg J$. When a fixed-point is reached, that is when no π-incompatible state is found and all states have their successors within the set of already reachable states, the intersection of all the constraints onto parameters is returned.

IM proceeds by iterative state space exploration, and its result comes under the form of a fully parametric constraint. By repeatedly applying the method, we are able to decompose the parameter space into a covering set of "tiles", which ensure a uniform behaviour of the system: it is sufficient to test only one point of the tile in order to know whether or not the system behaves correctly on the whole tile. This is known as the *behavioural cartography* [AF10]. Both IM and behavioural cartography are semi-algorithms; that is they are not guaranteed to terminate (i.e. to converge) but, if they do, their result is correct. However, compared with the more traditional Counter-Example Guided Approach (CEGA) for parameter synthesis [FJK08], IM is earlier to converge.

3.3 A Modular Framework for Modeling Real-Time Systems

Here we refer to a real-time system as a set of real-time tasks scheduled by a FP preemptive scheduler on a single processor. Our model of a real-time system consists of three kinds of components in Stopwatch Timed Automata: the task automata, the activation automata and the scheduler automaton. We refer to the composition of these automata through synchronisation as *system automaton*. The complete framework is shown in Figure 3.1. By default, clock variables in the model have rate 1.

Each task is modeled using a task automaton. Such a task automaton is shown in Figure 3.1a. Each task automaton contains two continuous clock variables c and d. Clock c counts the execution of the task and clock d counts the time passed since last job arrival. As we consider the generic activation pattern (periodic, sporadic or arrival curves), a new instance may be activated while the previous ones have not



(a) Task automaton

(b) Task activation automaton

(c) Scheduler automaton

Figure 3.1: The modeling framework for a real-time system

yet completed. Hence, there can be several active jobs from the same task at the same time. A discrete variable [1] N is used to count the number of simultaneous active instances for the task.

Initially, a task is in location Idle. The synchronisation label arrival_event notifies that a new instance from this task is activated and triggers a transition to a committed location ActEvent, where time elapsing is not allowed. The label arrival is used between a task and the scheduler. The task will then go to location Waiting and wait there for the scheduler's decision whether to occupy the processor. If a task is the highest priority among all active tasks in the system, the scheduler will send dispatch to trigger the transition from Waiting to Running. While a task is in Running, the scheduler can revoke the processor for a higher priority task through the synchronisation label preemption.

The clock d always progresses and the execution time clock variable c is stopped if a task is waiting. When a task is waiting for the processor or running on the processor, to react to new activations, it will non-deterministically choose to increase the counter N of active instances by 1. When a job misses its deadline ($d = D$) before completing its execution, it will go to DeadlineMissed. When a task finishes its execution ($c = N \cdot C$), it will go back to the initial location Idle.

There can be many different activation patterns for a task, such as periodic, sporadic or according to arrival curves. We only require that the activation automaton synchronises with the task automaton on label arrival_event. As a demonstration, Figure 3.1b shows the activation model for a periodic task. Every period T, the automaton sends the signal arrival_event to inform the arrival of a new job.

[1] Discrete variables are not part of the original LHA formalism, but can be seen as syntax sugar to increase the number of discrete states (locations). Such discrete variables are supported by most tools for LHA.

In this chapter, we assume tasks are scheduled according to a FP preemptive scheduler. The scheduler automaton synchronises with the tasks and decides which task will occupy the processor at each time. The structure of the automaton is completely fixed given a number of tasks.

Figure 3.1c shows a scheduler for two tasks. The scheduler automaton can be expanded in a similar form to deal with a task set with more tasks. In the scheduler automaton, the label arrival, dispatch, preemption and end are the same as in task automaton; we append a label with index i, e.g. end_i, to denote that this label synchronises with task i. The convention we use for naming the location encodes the status of the tasks: Rtx means that the task τ_x is running; Atx means that task τ_x is just activated; Wtx means that τ_x is waiting; Et is saying the task just finished execution. For simplicity, along with the transition from Rt1Wt2 to At1Rt2 there are two synchronisation labels.

3.4 Applying the Parametric Analysis

3.4.1 Convergence problem

By configuring certain constants in the model by Figure 3.1 as unknown parameters, we obtain a parameterised model and parametric analysis algorithms as IM can be applied. However, we first show that the application of the IM to a system with parametric task activations does not yield satisfactory results. Consider a task set with two periodic tasks $\tau_1 = (31, T_1, T_1)$ and $\tau_2 = (49, T_2, T_2)$ with implicit deadlines. If we apply IM with initial values $T_1 = 60$ and $T_2 = 120$, respectively, the final constraints obtained will be $T_1 = 60$ and $T_2 = 120$. That is, the result produced by IM is a single point, the initial valuation.

Such a result is caused by an important property of the scheduling problem. The IM synchronises a constraint that delimits the valuations of parameters that result in the same traces as the initial valuation. The schedule generated by a set of periodic real-time tasks is itself periodic with the hyperperiod H. In particular, the sequence of scheduling events (i.e., synchronisation labels in Figure 3.1) repeats itself every H, and different H will result in different traces of task execution. When task periods are unknown parameters, and since lcm is highly non linear, a small variation on one period can cause very large variations in the hyperperiod. For example, consider the two previous tasks with initial valuation of task periods $T_1 = 60$ and $T_2 = 120$, respectively. Their hyperperiod is 120. When we increase T_1 to 121, the hyperperiod becomes 7260. Clearly, in the second case, the traces are much longer and contain many more events. This explains why IM only converges to the initial valuation. Similarly, if we apply counter-example guided approach to find parameter valuations such that the DeadlineMissed location is not reached, the parameter synthesis cannot even terminate (in reasonable time and memory cost).

Of course, things become even more complex when considering generic arrival patterns. The next part solves this convergence problem by exploiting the well-known concept critical instant from classic scheduling theory.

3.4.2 An improved model of the system

As we just discussed, it is infeasible to apply IM directly to a system model with parametric arrival patterns. We will try to avoid this situation by adapting the system automaton (Figure 3.1) by exploiting the concept of critical instant.

For FP scheduling of periodic or sporadic tasks in a single processor, it is possible to define a *critical scenario*, which is the situation that arises when all tasks are simultaneously activated (critical instant) and every task τ_i generates subsequent jobs as soon as it is allowed. According to the work by Liu and Layland [LL73], the WCRT of a task can be found in the busy period (i.e. interval in which the processor

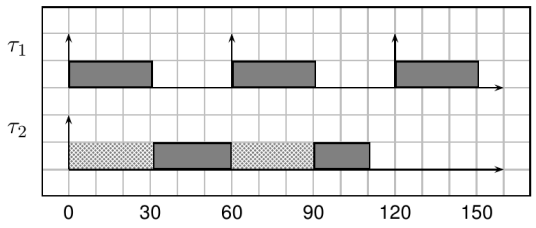


Figure 3.2: Schedule of the first busy period of the example task set

is continuously busy) that starts at the critical instant.

This means that, if we want to check the schedulability of a set of periodic or sporadic tasks, it is sufficient to activate all tasks at time zero and check that if there is deadline miss in the first busy period. Therefore, as soon as the processor becomes idle we can stop the search.

In the system automaton as in Figure 3.1, each trace corresponds to a possible schedule of the task set. However, we now know that to check the schedulability of a task set, it is sufficient to analyse traces starting from the critical instant till the first idle time in the processor. So, we adapt the system automaton as follows.

- The task activation automaton is required to release its first job at time 0 and it will emit the subsequent jobs as fast as possible.
- In the scheduler automaton, after all tasks complete their execution, instead of going back to Idle, it will transit from Et to a new location Stop, where there is no outgoing transition.

The first point is used to simulate the worst-case behaviour of tasks starting from the critical instant. Rather than going to Idle and waiting for new task releases, the scheduler automaton (also the system automaton) simply stops. We call this adapted scheduler model as *idle-time scheduler automaton*.

The idle-time scheduler automaton actually simulates the longest busy period, which starts from the critical instant and ends at the first idle time of the processor. The length of this busy period depends both on the execution time and activation periods of the tasks. However, its dependence from the periods is not as strong as the hyperperiod. Let us consider again the previous set of two periodic tasks $\tau_1 = (C_1 = 30, D_1 = T_1 = 60)$ and $\tau_2 = (C_2 = 49, D_2 = T_2 = 120)$. The schedule for the longest busy period is shown in Figure 3.2. Task τ_1 executes twice before the first instance of τ_2 can complete.

The length of the busy period in this case is $1C_1 + C_2 = 111$. By doing some simple calculation, it is easy to see that changing T_1 in to any value in $[56, 79]$ does not change the sequence of events in the busy period: in facts, for any value of T_1 in that interval, τ_1 will still execute two times before the first instance of τ_2 completes. Also, changing T_2 to any value $T_2 \geq 112$ does not change the busy period.

Hence, we can apply IM on the new model and avoid the convergence problem as in Section 3.4.1. Let us assume $T_1 \in [40, 120]$, $T_2 \in [80, 200]$ and let us apply the behavioural cartograph to obtain the constraint space of T_1 and T_2 that keeps the task set schedulable. The result in given in Figure 3.3 in a graphical form. The red part (on the left side) is the constraint space on T_1 and T_2 in which τ_2 misses deadline, whereas the green part (on the right side) is where no deadline is missed.

When applying the behavioural cartography to the model with idle-time scheduler automaton, there may exist a combination of parameter valuations that cause the system go into overload, i.e. there will be no idle time in the schedule.

To solve this case, we put an upper bound on the maximal depth of the traces computed by IM. This bound is always computable in case of periodic or sporadic tasks, and corresponds to computing an upper bound to the time a deadline miss will happen. A method for computing such a bound can be built by using the concept of *demand bound function* [BRH90].

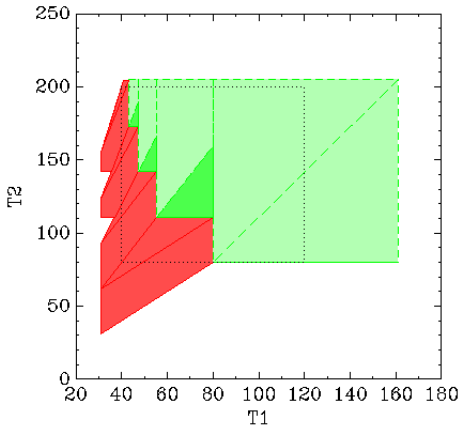


Figure 3.3: Constraints on T_1 and T_2 obtained by the behavioural cartography

3.5 Applicability of the Idle-Time Scheduler

It is possible to prove that the concepts of critical instant and the maximal busy period are valid also when considering tasks activated by generic arrival curves. In particulr, the critical scenario corresponds to the time instant in which all tasks are activated with their initial burstness (critical instant), and their successive instances arrive as soon as possible without violating their arrival curves. Then, the WCRT can be found in the busy period starting at the critical instant and corresponding to the critical scenario. Therefore, we can apply idle-time scheduler also for generic arrival curves. Let us assume a periodic arrival curve of the form as in Equation (2.1).

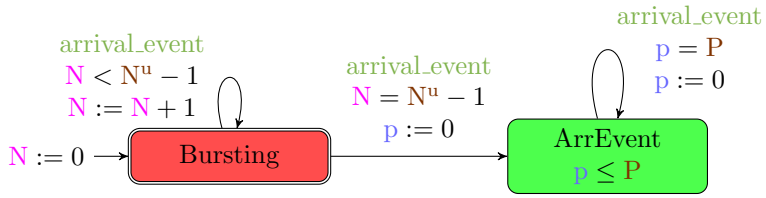


Figure 3.4: Arrival curve automaton

In Figure 3.4 we show the corresponding Stopwatch Timed Automaton model for a periodic arrival curve. Initially, the arrival curve automaton is in a committed location Bursting with $N = 0$, where N is a discrete variable counting the number of initial requests. The automaton emits N^u activations for a task within 0 time elapsing and then move to location ArrEvent where it starts behaving as a periodic activation automaton as in Figure 3.1b, and produces activation events every P.

For other different task models there may be no critical instant. For example, when considering periodic tasks with initial offset different from zero, there is no worst-case scenario in the schedule. Instead, it is necessary to analyse all busy periods in the interval $[0, 2H + \Phi_{max}]$, where Φ_{max} is the largest initial offset [LW82].

Given a task set $\mathcal{T}$ of periodic tasks with offsets, we can build a task set $\mathcal{T}'$ that contains the same tasks with the same parameters except that their initial offsets are all set to zero. In this case, if $\mathcal{T}'$ is schedulable, then also $\mathcal{T}$ is schedulable. However, the reverse does not hold. Therefore, it is possible to perform a parametric analysis of $\mathcal{T}'$ using idle-time scheduler, and the set of values of the unknown parameters produced by the analysis is a subset of all valid parameters for the original task set $\mathcal{T}$.

3.6 Toward Timed Interfaces

For complex distributed real-time systems, a component-based methodology may help reduce the complexity of the design and analysis phases. We define a distributed real-time system as a set of real-time components. Each component runs on a dedicated single processor node, and all components are connected to each other by a local network. A component consists of a *provided interface*, a *required interface*, and an *implementation* (see e.g. [Has12]). In the following, we describe our notions of the timed interface.

The provided interface is a set of methods that a component makes available to other components of the system. Each method is characterised by: (i) the method signature, which is the name of the method and the list of arguments, and (ii) a worst-case activation pattern, which describes the maximum number of invocations the method is able to handle in any interval of time. We will describe the worst-case activation pattern by an arrival curve. The semantic of invocation of a method can be synchronous (the caller waits for the method to be completed) or asynchronous (the caller continues to execute without waiting for the completion of the operation).

The required interface is a set of methods that the component requires for carrying out its services. Each method is characterised by its signature and a worst-case invocation pattern.

The implementation of a component is the specification of how the component carries out its work. In our model, a component is implemented by a set of concurrent real-time tasks and by a scheduler.

A graphical representation of a component is shown in Figure 3.5. In this example, the component provides one single method in the provided interface (pictorially represented by the red rectangle), and does not specify any method in the required interface. The component is implemented by three tasks: τ_1 and τ_3 are time triggered (the green clocks in the picture), whereas τ_2 implements the method in the provided interface, and hence it is triggered by invocations from external clients.

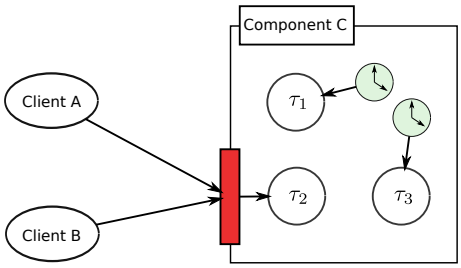


Figure 3.5: A component with three tasks and one method in the provided interface.

In a *component-based* design methodology, components are independently designed and developed, and then *integrated* in the final system by connecting them together through their interfaces. It is clear that the interface specification plays an important role in this methodology: for a real-time component, the interface should contain not only the functional specification (i.e. method signature, constraints on its arguments, etc.) but also the *timed behaviour* of the component. In particular, we enhance the specification of the interface by adding parameters on the activation pattern and on the response delay of a method.

Given a component, its *provided interface* is thus defined as:

- a set of method signatures $m_1, m_2, \ldots$;
- a *parametric arrival curve* $\alpha_i(t)$ for each method m_i, which represents the activation pattern that the corresponding implementing task will receive;
- a *worst-case response time* D_i parameter for each method m_i.

Similarly, the *required interface* of a component is defined as:

- a set of method signatures $m_1, m_2, \ldots$;
- a *parametric arrival curve* $\alpha_i(t)$ for each method m_i that represents the activation pattern generated by this component;
- for every synchronous method call, a maximum allowed delay R_i in receiving the response.

Finally, the component is characterised by a *set of constraints* on the parameters: for all valuations of the parameters satisfying the constraints, the component is guaranteed to be correct both from the functional point of view (i.e. the component produces correct values) and from the timing point of view (i.e. all tasks complete before their deadlines, and all provided functions return their values within the desired maximum response delay).

The road to realise such a component-based design methodology is long and many theoretical and practical problems need to be solved before the methodology can be used in practice. One important problem is how to compute the set of constraints that define the correct behaviour of a component. In the process of designing and analysing a component in isolation, it is necessary to use parametric arrival curves for describing the activation patterns for tasks, and parametric deadlines for bounding their response times. Performing a parametric analysis aims at deriving a set of constraints for these parameters that make the component schedulable. During integration, the correctness of the system is checked by intersecting the constraints of the communicating components to see if there is some feasible assignment of parameters that makes all components schedulable.

For the sake of simplicity, we focus only on the *provided interface* of a component; that is, we investigate on the parametric analysis of a component with respect to the patterns of activations. The analysis of the required interface is the subject of future work.

3.7 The Timed Interface

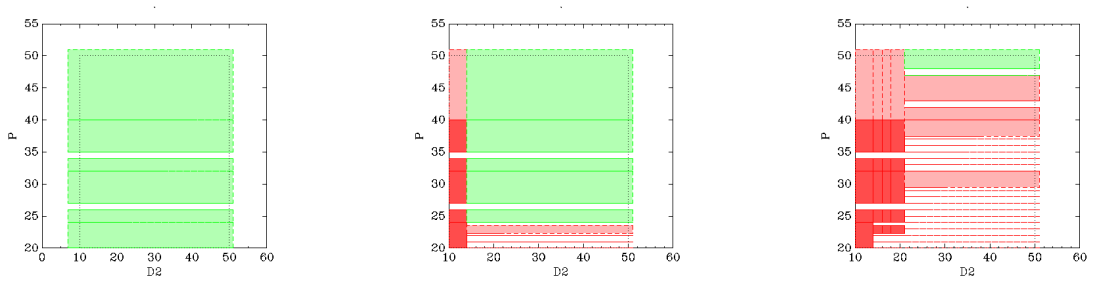
Now we show how it is possible to define a *timed interface* of a real-time component using parametric analysis.

Consider the system of Figure 3.5: it consists of three tasks τ_1, τ_2 and τ_3 running on a single processor with FP preemptive scheduling. τ_1 and τ_3 are periodic tasks with $\tau_1 = (C_1 = 2, D_1 = 8, T_1 = 8)$ and $\tau_3 = (C_3 = 20, D_3 = 50, T_3 = 50)$. Task τ_2 has $C_2 = 5$ and implements the method provided in the interface. We assume that this component is linked to a local networks, and task τ_2 receives the requests from clients running on other nodes of the network. We would like to know how many clients can ask requests to the system, with which frequency, and the maximum delay that is going to pass from the request to response. Therefore, we need to study the possible activation patterns of task τ_2 and its WCRT. For modeling te activation patterns, we use a parametric arrival curse. For example, $N^u = 2$ and $P = 100$ means that we can connect at most 2 independent clients, and that between any two successive requests after the first two there must at most 100 units of time.

Both N^u and P are parameters we are going to synthesise with our parametric analysis. Another parameter is the delay (deadline) D_2 of τ_2. We are interested in the parameter space that guarantees all the tasks schedulable.

First, we construct the activation automaton for $\alpha(t)$ as in Figure 3.4. Following the method described in Section 3.4.2, and using the idle-time scheduler automaton, we then compose the final automaton.

Given that $C_2 = 5$, it is easy to see that the burst (N^u) of the arrival curve automaton cannot be larger than 3, otherwise τ_3 will be doomed to miss its deadline, because $D_3 < C_3 + 4C_2 + 5C_1$.



(a) $N^u = 1$ (b) $N^u = 2$ (c) $N^u = 3$

Figure 3.6: Parameter space (green) for N^u, P and D_2

$N^u = 1$	when $(20 \leq P \leq 26) \vee (27 \leq P \leq 34) \vee (35 \leq P \leq 50) \rightarrow D_2^{min} = 10$
$N^u = 2$	when $(24 \leq P \leq 26) \vee (27 \leq P \leq 34) \vee (35 \leq P \leq 50) \rightarrow D_2^{min} = 14$
$N^u = 3$	when $(P = 47) \vee (48 \leq P \leq 50) \rightarrow D_2^{min} = 21$

Table 3.1: The final interface

Additionally, we assume P and D_2 lie in the following intervals:

$$P \in [20, 50], D_2 \in [10, 50].$$

N^u is a discrete parameter that must be treated separately from P and D_2. Our strategy is to instantiate N^u with 1, 2 and 3 individually and apply IM to each case in order to synthesise constraints over P and D_2 that keep the system schedulable. The resulting parameter space for the three cases are visualised in Figure 3.6.

We can use these values to build a timed interface specification for the component.

- The number of distinct independent clients that can be connected to the service must respect the constraint $1 \leq N^u \leq 3$.
- Depending on the number of clients, the relationship between the minimum period P and worst-case response time D_2 is specified in Table 3.1.

Reducing the number of regions As it is possible to see in Figure 3.6 and Table 3.1, the parameter space obtained by IM consists of a set of disjoint tiles. Each tile is a convex region and the resulting interface is the union of (maybe a large number of) these convex regions. Such an interface may not be easy to use due to the large number of disjoint regions.

To obtain a more portable interface, we can make an effort from two different directions.

In some cases, it is possible to perform a "merge" operation between tiles, as explained in [AFS13], in order to reduce the number of convex regions composing the final interface. Two convex regions are mergeable if their convex hull equals to their union. Given tiles returned by IM, we repeatedly replace mergeable tiles with their union till there are no mergeable tiles. Moreover, another parameter synthesis algorithm based on reference point has been developed in [ALNS15] such that it guarantees the resulting constraint preserves the system to have the same reachability property with respect to certain location (e.g. DeadlineMissed) and compared with IM, it can return a larger constraint consisting of more parameter valuations.

On the other hand, according to the result from real-time scheduling theory, single processor FP scheduling is fully sustainable [BB06]. That means, given a schedulable task set under FP scheduling, if we increase the value of a task's inter-arrival time or relative deadline, the resulting task set is schedulable by FP scheduling. By applying this result, the IM can be applied more efficiently: as long as a reference point is decided by IM as a good (bad) point for schedulability, all points with a larger (smaller) value with respect to any parameter will be also a good (bad) point; thus, the number of disjoint tiles can be dramatically reduced.

3.8 Conclusion

In this chapter, we have presented a Stopwatch Timed Automaton model for a real-time system scheduled by FP on a single processor. We have shown how to perform a parametric analysis using IM with a specific model of the scheduler that stops at the first idle time. Finally, we have shown how to use parametric analysis for the design and the specification of the interface of a real-time component.

On one side, parametric analysis based on formal models can deal with situations where the analytical approach cannot be applied. On the other side, to make the formal method feasible or efficient, we need to utilise classic results and domain knowledge from real-time scheduling. In our case, we apply the critical instant based worst-case scenario to improve the automaton model and force the parametric analysis to terminate, and we propose to use the sustainability property of FP scheduling to achieve more portable interface design.

Chapter 4

Component-Based Schedulability Analysis

Formal methods (e.g. Timed Automata or Linear Hybrid Automata) can be used to analyse a real-time system by performing reachability analysis on the model. The advantage of using formal methods is that they are more expressive than classic analytical models used in schedulability analysis. For example, it is possible to express state-dependent behaviour, arbitrary activation patterns, etc.

In this chapter we use the formalism of Linear Hybrid Automata (LHA) to encode a hierarchical scheduling system. In particular, we model a dynamic server algorithm and the tasks contained within, abstracting away the rest of the system, thus enabling component-based schedulability analysis. We prove the correctness of the model and the decidability of the reachability analysis for the case of periodic tasks. Then, we compare the results of our model against classic schedulability analysis techniques, showing that our analysis performs better than analytical methods in terms of resource utilisation. We further present two case studies: a component with state-dependent tasks, and a simplified model of a real avionics system. Finally, through extensive tests with various configurations, we demonstrate that this approach is usable for medium-size components.

The content of this chapter is based on [SLS+14]. Though Chapter 3 also mentions the concept of "component", its emphasis is in interface design, and in this chapter the component is used for temporal isolation and the main result is on testing the schedulability of real-time tasks inside a component independently from other components in the system.

4.1 Introduction

The complexity of modern embedded real-time applications, like automotive and avionics systems, is steadily increasing. Until recently, complexity was addressed by using physical separation: each different functionality was implemented by a different application module on a different ECU (Electronic Control Unit) and all ECUs were connected by a real-time control network.

The pressure to reduce the design cost and the number of ECUs is forcing developers to integrate different applications on the same platform. The IMA (Integrated Modular Avionics) [WW07, Spe91] is a set of standard specifications for simplifying the development of avionics software; among other requirements, it allows different independent applications to share the same hardware and software resources [ARI96].

To avoid the interference among independently developed applications that share the same processor, the underlying RTOS (Real-Time Operating System) must support the concepts of *temporal partitioning*

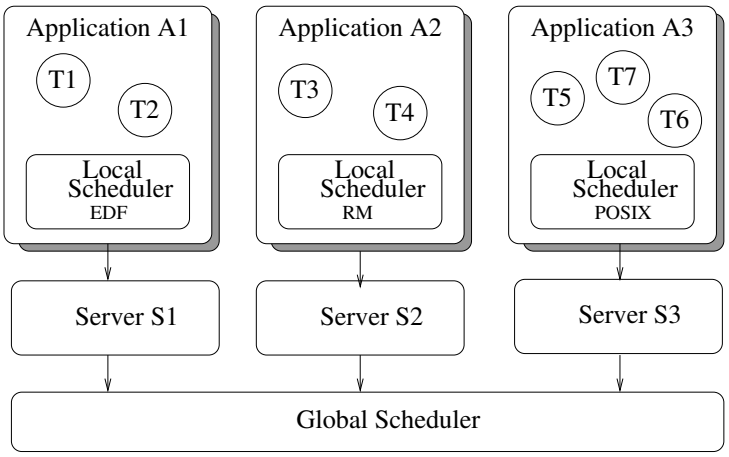


Figure 4.1: An example of hierarchical scheduling system.

and *hierarchical scheduling* [FM02, SL03, LB04]. Hierarchical scheduling consists in using two (or more) levels of scheduling: the global one performs the temporal partitioning among the applications; whereas the local ones are specific for each application and dedicate which task to execute. In Figure 4.1 we show a pictorial representation of a hierarchical system consisting of three applications that coexist in the same single processor system, each one with its own scheduler. Each application and its corresponding local scheduler are "wrapped" into an entity that we denote as *scheduling server* (or simply *server*) which acts as a mediator between the global scheduler and the application. The global scheduler "sees" the entire application as it were a single task to be scheduled according to its specific scheduling parameters; the application "sees" the platform on which it is executing as a virtual processor of slower speed.

Therefore, the combination of temporal partitioning and hierarchical scheduling makes it possible to define a virtual processor for each application, and to perform schedulability analysis on the virtual processor rather than on a single dedicated processor.

If the applications running on a system are independent of each other, then it is possible to analyse each of them in isolation; in fact, the ability of an application to meet its deadlines depends on the worst-case computation times and the arrival patterns of its tasks, and on the temporal partition that the global scheduler (and the server) allocate to it, but it *does not* depend on the presence of other applications in the system.

Such a property enables *component-based* schedulability analysis, a research topic largely investigated in recent years.

4.2 State of the Art

The ARINC 653 standard [ARI96] defines temporal partitioning for avionics applications. The global scheduler is a simple Tim Division Multiplexing (TDM), in which the time is divided into slots and each slot is assigned to one application. Besides TDM, more dynamic time partitioning algorithms are possible: for example the periodic resource model [SL03] and the periodic server [LB04].

Dynamic server algorithms have advantages over TDM. First of all, the temporal interface of a periodic server consists only of two parameters: the budget Q and the period P; the server guarantees that the application will receive Q time units every P, but unlike the TDM, it does not specify at which precise instants the application will receive the allocation. This means that, once the application has been guaranteed feasible on a server with certain parameters Q and P, during the integration phase the designer has much more freedom in the allocation of the budget. The second advantage is that a dynamic

server algorithm can better take advantage of the dynamic behaviour of the application and adapt itself at run-time to different conditions.

Hierarchical scheduling and component-based real-time scheduling algorithm have been studies extensively in the past years. Feng and Mok [FM02] proposed the resource partition model and schedulability analysis for it. Shih and Lee [SL03] introduced the concept of temporal interface and the periodic resource model. Lipari and Bini [LB04] proposed the periodic server model to abstract many different temporal partitioning algorithms, and an algorithm to compute the values of the parameters to make the application schedulable. Davis and Burns [DB05] proposed a method to compute the response time of tasks running on a local fixed priority scheduler when task periods are synchronised with the server period.

In [APN11], a formal model of hierarchical scheduling systems using Timed Automata has been proposed. The goal of the authors is to verify the correctness of the two-level scheduler and generate C code for the scheduler and the tasks. Moreover, their analysis is *global* in the sense that they verify the correctness of the whole system rather than a single application. Instead, we aim at component-based schedulability analysis of a single application. Another formal model of hierarchical scheduling using parametric Timed Automata has been proposed in [AFKS12, FLMS12]. The authors restrict themselves to a TDM global scheduler and perform a global analysis (rather than a component-based one).

A component-based analysis of hierarchical real-time systems is proposed in [CLPV11, CPV13]. The authors use the model of Preemptive Timed Petri Nets (pTPN), to model a hierarchical systems, and perform analysis of independent applications. They show that component-based analysis considerably reduces the complexity of analysing a system. The main difference with the work here is that they model a TDM global scheduler, whereas we model a dynamic periodic server algorithm.

4.3 The Hierarchical System

In this chapter we assume that an application is a set of periodic real-time tasks with explicit initial offsets $\mathfrak{A} =< \tau_1, \tau_2, \ldots, \tau_n >$. The system consists of a set of applications to be scheduled using hierarchical scheduling and temporal partitioning on a single processor system. We assume that all applications are independent of each other thus that we can analyse them in isolation.

Each application is executed upon a *virtual processor platform*, which is provided by a *server*. In this chapter we consider the periodic server proposed in [LB04]. Each server is assigned a budget Q and a period P, and the global scheduler guarantees that the application will receive Q units of execution time every P time units. The global scheduler performs Earliest Deadline First (EDF) among the servers: each server is considered as a periodic task with period and relative deadline P, and worst-case computation time Q. Therefore, it must hold true that $\sum_i \frac{Q_i}{P_i} \leq 1$.

As for the local scheduler, we assume the Fixed Priority (FP) preemptive scheduler.

4.4 Server Algorithm

In this section we present the server algorithm that is used to provide the temporal partition necessary for an application to execute. We use the same algorithm proposed in [LB04], which is a particular case of the Constant Bandwidth Server [AB98]. We summarise the algorithm here for convenience.

A server S is assigned two parameters: Q is the server maximum budget, and P is the server period. In addition, the server maintains three internal variables: q represents the *current remaining budget*, and d is the current *scheduling deadline*, and an internal state which can be one of the following:

- Idle: the initial state; it represents the situation in which no task is active in the application;

- Active: when there is at least one active task, but the server is not executing because other servers (for other applications) with earlier scheduling deadlines have been selected by the global EDF scheduler;
- Executing: when the server has been selected by the global EDF scheduler, and it is running an application task;
- Recharging: when there is at least one active task in the application, but the server cannot execute because the current budget is zero;
- Empty: when there is no active task, but the server has already consumed part of its budget, so it has to wait before it can become Idle again.

The server variables and the server state are updated according to the following rules.

1. Initially, $q = 0, d = 0$ and the server is Idle.
2. When a task is released at time t, if the server is Idle, then $q := Q$ and $d := t + P$, and the server becomes Active; if the server is already Active, then nothing needs to be done.
3. At any time t, the global scheduler selects an Active server, and a task inside it will be chosen to run subject to local scheduling policy. The chosen server moves to the Executing state.
4. While some task in the server is running, the current budget q is decremented accordingly.
5. The global scheduler can preempt a server to execute another server. The preempted server moves back to the Active state.
6. If q reaches 0 and some task has not completed execution, then the server will move to Recharging and be suspended till time d. During the suspended interval, it cannot be chosen by the global scheduler. At time d, q is recharged to Q and d is set to $d + P$ and the server moves to the Active state.
7. When, at time t, the last task in the server has finished its execution, if $t \geq d - q\frac{P}{Q}$, the server becomes Idle; otherwise, it remains Empty, and will become Idle at time $d - q\frac{P}{Q}$, unless another task arrives before that time point.

The global scheduler performs the Earliest Deadline First policy using the scheduling deadlines of the servers. The following two results are direct consequences of Theorem 1 and Lemma 1 of [AB98]:

Theorem 1. *Consider a system consisting of n servers, $\{S_1, \ldots, S_n\}$, with $S_i = (Q_i, P_i)$, such that $\sum_{i=1}^{n} \frac{Q_i}{P_i} \leq 1$. Then, no server misses its scheduling deadlines.*

Theorem 2. *Given a server $S_i = (Q_i, P_i)$, let t_s be an instant in which the server moves from the idle state to the active state, and let t_f be the first instant after t_s such that the server becomes idle again. Then the server receives in interval $[t_s, t_f]$ an amount of execution time Δ_{exe} which is bounded by:*

$$\left\lfloor \frac{t_f - t_s}{P_i} \right\rfloor Q_i \leq \Delta_{\mathsf{exe}} \leq \left\lceil \frac{t_f - t_s}{P_i} \right\rceil Q_i$$

4.5 Periodic Server Model in LHA

In this section, we introduce a LHA for modeling the periodic server algorithm described in Section 4.4. In the model, we need to stop the clocks, since our servers and tasks can be preempted. Also, we need to use arbitrary linear constraints on clock variables. For convenience, we decided to rely on the more general model of LHA rather than restrict ourselves to Stopwatch Timed Automata.

If we want to precisely model a system of n applications $\{\mathfrak{A}_1, \ldots, \mathfrak{A}_n\}$, each one served by a server S_i with parameters (Q_i, P_i), we have to build:

- n automata, one per server;
- one automaton for modeling the global EDF scheduler;
- one automaton per task;
- and finally, one automaton per local FP scheduler.

The final system can be represented by the parallel composition of all such automata. However, this approach has two main inconveniences: first of all, it is specific for one single system, and it would be necessary to build a new model for each different system; second, the resulting automaton is very complex even for a small number of applications and tasks (state-space explosion problem).

We assume that applications are independent of each other, thus we can analyse each application in isolation. It is important to underline that such assumption is basically the same used in avionics real-time applications that have been designed according to the IMA architecture: tasks belonging to different applications can only communicate through non-blocking communication primitives. Therefore, we can use appropriate abstractions to build the model of one single server: in particular, we will abstract away the presence of the other servers and the global scheduler.

We make a one-to-one correspondence between states of the algorithm and locations of the LHA. In particular, we use:

- one location for each *state* of the algorithm;
- two different continuous clock variables: variable x represents the consumed budget, whereas variable y represents the time passed since the beginning of the server period.

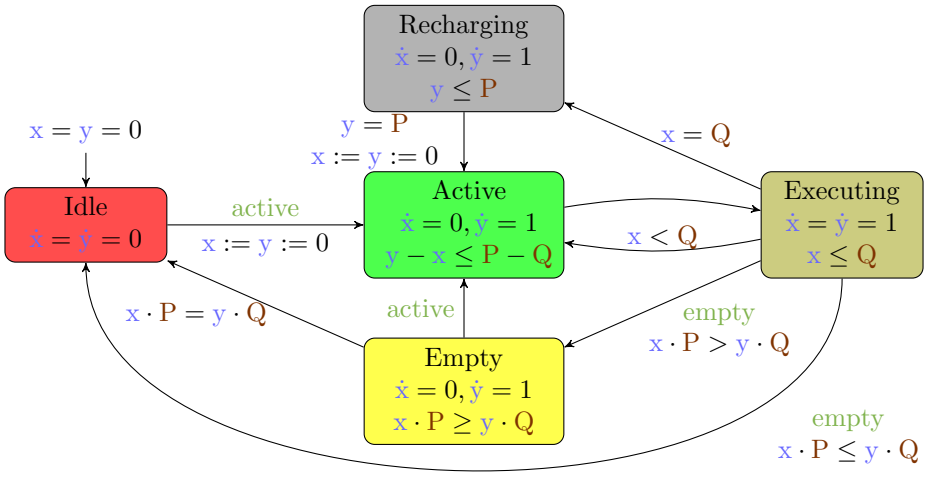


Figure 4.2: The Server automaton.

The Server linear hybrid automaton is depicted in Figure 4.2. Idle is the initial location. The application tasks served by this server are modeled with two synchronisation labels: active notifies a task's activation and empty means that the last task in the server has finished its execution.

If a task arrives when the Server automaton is in Idle, the model goes to location Active. The transition from Active to Executing happens when the global scheduler picks the server to execute. The reverse transition from Executing to Active models server preemption. Notice that these two transitions have no synchronisation labels because we want to abstract away the presence of other servers in the system and of the global scheduler.

Since we abstracted away the description of the global EDF scheduler and of the other servers, we need to add some constraints to guarantee that the model behaves correctly. We take for granted that Theorem 1 holds, and that therefore the server will meet all its scheduling deadlines. We impose such property by adding invariant $y - x \leq P - Q$ to location Active. This invariant states that, while in Active, there is still enough time to complete the execution of Q units before the end of the period. Therefore, no later than the time when $y - x = P - Q$ the automaton has to move to location Executing.

When the currently used budget reaches $x = Q$, the Server automaton moves from Executing to Recharging. It will stay in location Recharging until the start of a new period, at which point the current consumed budget is reset to $x := 0$. Location Empty models Rule 7 of the algorithm in Section 4.4: if the server tasks finish executing too early ($x \cdot P \leq y \cdot Q$), then the automaton directly moves to location Idle. If it is too late ($x \cdot P > y \cdot Q$), the automaton first moves to location Empty where it waits for the time y to reach the appropriate value before moving to Idle and resetting the model.

4.5.1 Proof of correctness

We now prove that the proposed Server automaton correctly models the server algorithm. In particular, we are going to prove that, under the assumption of Theorem 1, the automaton also respects Theorem 2.

Theorem 3. *Let* Server *be an automaton with parameters* (Q, P) *that models a dynamic periodic server, and let* t_s *be an instant in which the automaton moves from location* Idle *to location* Active. *Let* t_f *be the first instant after* t_s *such that the automaton enters again location* Idle. *Let* $\Delta_{exe}(t_s, t_f)$ *be the total amount of time that the server spends in location* Executing *in interval* $[t_s, t_f]$. *Then,*

$$\left\lfloor \frac{t_f - t_s}{P} \right\rfloor Q \leq \Delta_{\text{exe}} \leq \left\lceil \frac{t_f - t_s}{P} \right\rceil Q$$

Proof. To prove the theorem, we start by adding an extra Error location, and a transition from Executing to Error with guard:

$$x < Q \wedge y = P.$$

To make the expression more compact, we denote variable valuations subject to a constraint like $x < Q \wedge y = P$ to be the form of $\{x < Q, y = P\}$.

Let us observe that variable x is incremented only when the automaton is in location Executing. Also, x is reset to 0 after it reaches its maximum value Q (transition from Execution to Recharging), and when y reaches P. Hence, $\Delta_{exe} = n_r Q + x$, where n_r is the number of times the automaton goes through location Recharging in interval $[t_s, t_f]$. If the automaton does never reach location Error, then $n_r = \lfloor \frac{t_f - t_s}{P} \rfloor$ and both the lower bound and the upper bounds of Δ_{exe} are trivially true.

Therefore, it remains to be proved that Location Error is unreachable for any values of Q and P with $Q \leq P$. We do this by computing symbolically the reachable valuations of all variables in all locations, by using fixed point iterations: we start by the initial values of the variables in all locations, and we apply the time elapsing operation. It is trivial to check Locations Idle and Recharging. For other locations, we

apply the iterative method used in [HPR97].
First step,

$$\begin{aligned} R_{Act} &= ((\{x = y = 0\}) \nearrow \{\dot{x} = 0, \dot{y} = 1\}) \cap \{y - x \leq P - Q\} \\ &= \{x = 0, y \leq P - Q\} \end{aligned}$$

$$\begin{aligned} R_{Exe} &= ((R_{Act} \cap x \leq Q) \nearrow \{\dot{x} = \dot{y} = 1\}) \cap \{x \leq Q\} \\ &= \{0 \leq x \leq Q, x \leq y \leq x + P - Q\} \end{aligned}$$

$$\begin{aligned} R_{Emp} &= ((R_{Exe} \cap \{xP \geq yQ\}) \nearrow \{\dot{x} = 0, \dot{y} = 1\}) \cap \{xP \leq yQ\} \\ &= \{0 \leq x \leq Q, x \leq y \leq x\frac{P}{Q}\} \end{aligned}$$

Second step,

$$\begin{aligned} R_{Act} &= (\{0 \leq x \leq Q, x \leq y \leq x + P - Q\} \nearrow \{\dot{x} = 0, \dot{y} = 1\}) \cap \{y - x \leq P - Q\} \\ &= \{0 \leq x \leq Q, x \leq y \leq x + P - Q\} \end{aligned}$$

$$\begin{aligned} R_{Exe} &= (\{0 \leq x \leq Q, x \leq y \leq x + P - Q\} \nearrow \{\dot{x} = \dot{y} = 1\}) \cap \{x \leq Q\} \\ &= \{0 \leq x \leq Q, x \leq y \leq x + P - Q\} \end{aligned}$$

$$\begin{aligned} R_{Emp} &= (\{0 \leq x \leq Q, x \leq y \leq x + P - Q\} \nearrow \{\dot{x} = 0, \dot{y} = 1\}) \cap \{xP \geq yQ\} \\ &= \{0 \leq x \leq Q, x \leq y \leq x\frac{P}{Q}\} \end{aligned}$$

After two steps we have already found the fixed point for R_{Exe} and R_{Emp}. With the third iteration, we compute the fixed point also for R_{Act}. The final results are:

$$R_{Act} = \{0 \leq x \leq Q, x \leq y \leq x + P - Q\}$$

$$R_{Exe} = \{0 \leq x \leq Q, x \leq y \leq x + P - Q\}$$

$$R_{Emp} = \{0 \leq x \leq Q, x \leq y \leq x\frac{P}{Q}\}$$

The reachable valuation for Error is simply the intersection of R_{exe} with $\{x < Q, y = P\}$ which is empty. Therefore, location Error is unreachable. □

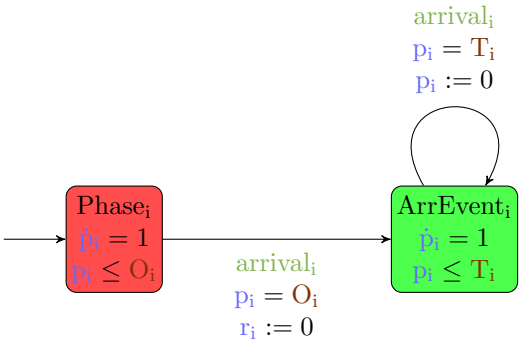
4.6 Schedulability Analysis in the Hierarchical System

In this section, we use the Server automaton to perform the schedulability test of an application on a periodic server. Without loss of generality, we adapted the encoding used in Chapter 3 to show how to combine already existing schedulability model with our LHA model, in order to check if a task set in a server will miss its deadline.

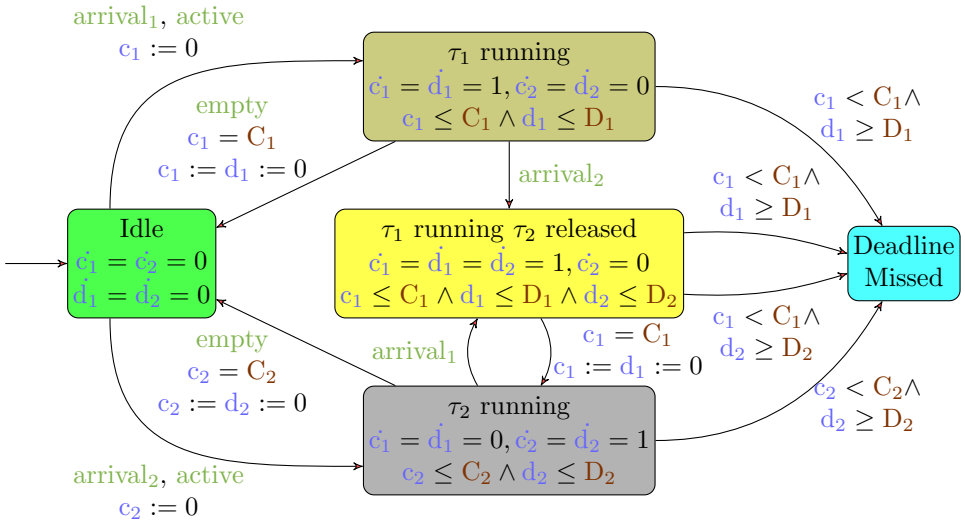
4.6.1 Scheduler automaton

We now show how to encode an application with a FP local scheduler using an example with two tasks $\{\tau_1, \tau_2\}$. In Fig. 4.3 we show the Scheduler automaton that encodes the FP scheduler along with the task execution, and the automaton $\mathsf{Arr_i}$ that models the activation of the two tasks.

Let's start from the latter: each task arrival patterns is modeled with a timed automaton $\mathrm{Arr_i}$ with



(a) The activation automaton Arr_i



(b) The scheduler automaton Scheduler

Figure 4.3: Model of a FP scheduler for two periodic tasks τ_1, τ_2.

just two locations, and one clock p_i which is always increasing. The first transition from location Phase to ArrEvent models the first release time at the task offset; the second transition is a loop from ArrEvent to itself that models subsequent releases. It is easily possible to model different arrival patterns by simple changing the corresponding arrival automaton.

For simplicity, the Scheduler automaton here is actually the parallel composition of the task automaton and the scheduler automaton as in Figure 3.1. In the Scheduler automaton, we use two kinds of clock variables: *executing variables*, such as c_1 and c_2, for recording a task's accumulating execution time; and *deadline variables* (d_1 and d_2) for tracking if a task misses its deadline. The synchronisation label empty and active are the same as the in Server automaton. We accept there exist more than one synchronisation label on a transition in Scheduler automaton. Take the transition from Idle to τ_1 running, which has two labels arrival$_1$ and active on it, as an example. In order to trigger this transition, the Scheduler should first synchronise with Arr_1 through arrival$_1$, then (with no time elapsing) synchronise with Server through active.

Each location in the Scheduler automaton models a different state of the ready queue of the scheduler. Location Idle models an empty ready queue; location "τ_1 running" models the case in which only task τ_1 is active and running; location "τ_1 running τ_2 released" models the case in which the ready queue contains both τ_1 and τ_2, but τ_1 is running because it has the highest priority. Location "τ_2 running" models the case in which only τ_2 is active and running. Finally, location DeadlineMissed models the case in which one of the two tasks misses its deadline. Schedulability can be checked by performing a reachability

analysis for location DeadlineMissed.

Figure 4.3 only models the schedule of two tasks. Generating the model for more tasks can be done automatically by generating all possible configurations of the ready queue. This means that the size of the model is exponential in the number of tasks in the application. However, consider that in most practical cases, the number of tasks inside one application is limited to a few units. Also, component-based analysis abstracts away the rest of the system and hence it is much less complex than analysing the entire system as a whole, as shown in [CLPV11].

4.6.2 Hierarchical composition

The automata of Figure 4.3 models an application consisting of two tasks running on a single processor. We now describe how to compose such model with the LHA model of the server presented in Section 4.5.

Definition 2. *A **Hierarchical Scheduling Composition** of a task set with a periodic server is defined as the parallel composition of the server automaton* Server*, the scheduler automaton* Scheduler *and the task arrival automata* $\mathsf{Arr_1}, \ldots, \mathsf{Arr_n}$*:*

$$\mathsf{HSC} = \mathsf{Server} \times \mathsf{Scheduler} \times \mathsf{Arr_1} \times \ldots \times \mathsf{Arr_n}$$

with the following additional rule:

- *Let* $l \in \mathsf{Loc}(\mathsf{HSC})$ *be a location of the composed automaton, with* $l = (l_{Ser}, l_{Sched}, l_1, \ldots, l_n)$*, and* $l_{Ser} \in \mathsf{Loc}(\mathsf{Server})$*,* $l_{Sched} \in \mathsf{Loc}(\mathsf{Scheduler})$*, and* $l_i \in \mathsf{Loc}(\mathsf{Arr_i})$*, for all* $i = 1, \ldots, n$*. If* $l_{Ser} \neq$ Executing*, then the derivatives of all execution time variables are set to 0:* $\dot{c}_i = 0$*, for all* $i = 1, \ldots, n$*.*

4.6.3 Decidability

Once again, schedulability analysis can be encoded as a reachability analysis over automaton HSC. We now prove that such analysis is decidable for the case of independent periodic tasks.

Lemma 1. *Given a* HSC *automaton that models a set of periodic tasks* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *executing in a periodic server. The task set is schedulable if, and only if, location* DeadlineMissed *is unreachable in a time interval equal to* $[0, 2 \times lcm\{T_1, \ldots, T_n, P\} + \max\{O_i\}]$*, where* T_i *and* O_i *is the period and the initial offset of task* τ_i*.*

Proof. The proof uses a well-known result by Leung and Whitehead [LW82]: "A set of periodic tasks with deadlines less than or equal to the periods is schedulable if, and only if, there is no deadline miss in the interval $[0, 2H + \max\{O_i\}]$", where H denotes the hyper period. Intuitively, the reason is that the arrival pattern of a set of periodic tasks will repeat every hyperperiod after an initial transitory $2H + \max\{O_i\}$, so the schedule is periodic, and it is sufficient to check in the first periodic instance of the schedule.

Now we also take the periodic server into account. We are interested in finding a similar period such that not only task arrival patterns repeat themselves, but also all possible server behaviours will be identical. For finding the upper bound of the length of such an interval, we regard the server as a periodic task with period P, worst-case execution time Q and initial offset 0. Then we apply Leung and Whitehead test on the extended task set including original task set and the server. Thus, to check if a task misses its deadline in a server, it is sufficient to check all possible paths in the time interval $[0, 2 \times lcm\{T_1, \ldots, T_n, P\} + \max\{O_i\}]$. Notice that, thanks to non-determinism, the HSC automaton already models all possible generated schedules for the servers, therefore it covers all possible scheduling behaviours in the considered interval. □

From Lemma 1, it follows that our problem can be expressed as a problem of *reachability in bounded time*, in particular, we need to analyse the system and see if a DeadlineMiss state is reachable in $[0, 2 \times lcm\{T_1, \ldots, T_n, P\} + \max\{O_i\}]$.

Reachability analysis in bounded time has been proved to be decidable for a particular sub-class of LHA called *Rectangular Automata* [BDG+11]. Unfortunately, the HSC automaton does not fall in this class, since the server automaton in Figure 4.2 uses diagonal constraints as invariant in locations Empty and Active and as guard in the transitions between Empty and Idle, and between Executing and Idle.

In a nutshell, the problem is that in a LHA there can be an unbounded number of transitions in a finite interval of time. This effect is sometimes referred to as *Zeno effect* [AD94], as the distance between any two transitions can be made arbitrarily small, hence bounded time does not imply runs of bounded lengths. Using techniques from [BDG+11], we will now prove that reachability analysis can be performed on an HSC by only exploring paths of finite and bounded length, hence with a terminating algorithm.

We start by introducing a few necessary concepts and definitions that were used in [BDG+11].

Definition 3. *A* path *in HSC is a finite sequence of edges $e_1, e_2, \ldots, e_n$ such that the source location of e_{i+1} coincides with the destination location of e_i.*

A cycle *is a path where the destination location of e_n coincides with the source location of e_1. A cycle is* simple *if $\forall i \neq j$ the source location of e_i is different from the source location of e_j.*

A timed path *is a finite sequence of the form $\pi = \{(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)\}$, where $e_1, e_2, \ldots, e_n$ is a path in HSC, and $\forall i, t_i \in \mathbb{R}^+$. We can similarly define* timed cycles. *We denote by $\pi[i : j]$ the timed path $(t_i, e_i), \ldots, (t_j, e_j)$.*

A run *is a sequence $\rho = s_0, (t_1, e_1), s_1, (t_2, e_2), s_2, \ldots, (t_n, e_n), s_{n+1}$ such that s_i are states of HSC, $(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)$ is a timed path, and for each s_i there exist s'_i such that $s_i \xrightarrow{t_{i+1}} s'_i$ and $s'_i \xrightarrow{e_{i+1}} s_{i+1}$.*

The duration *of a run is defined as the total time spent executing the run: $\mathsf{dur}(\rho) = \sum_i t_i$.*

The length *of a run is defined as the number of discrete steps in the run: $\mathsf{len}(\rho) = n$.*

We now define the *contraction operator*, as in [BDG+11].

Definition 4. *Let $\pi = \{(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)\}$ be a timed path. Let j, j', k, k' be four positions in the path such that $1 \leq j \leq k < j' \leq k' \leq n$ and $e_j, \ldots, e_k$ and $e_{j'}, \ldots, e_{k'}$ two* simple identical cycles. *If such $1 \leq j \leq k < j' \leq k' \leq n$ exists then*

$$\begin{aligned}\mathsf{Cnt}(\pi) = \pi[1/j-1] \cdot (t_j + t_{j'}, e_j) \cdots (t_k + t_{k'}, e_k) \\ \cdot \pi[k+1 : j'-1] \cdot \pi[k'+1 : n]\end{aligned}$$

otherwise $\mathsf{Cnt}(\pi) = \pi$.

We also denote with $\mathsf{Cnt}^*(\pi)$ the repetitive application of Cnt to π until a fixed point is reached. Therefore $\mathsf{Cnt}^*(\pi)$ only contains one occurrence of each simple cycle.

Theorem 4. *The problem of reachability analysis in bounded time of any instance of the HSC automaton is decidable.*

Proof. We prove that if the target location is reachable within the time bound, then there is a *short* timed path (in terms of its number of locations) from the initial state to the target location. This is achieved by looking for loops that can be performed in arbitrarily short delay.

First of all, let us focus on the scheduling automaton, an example of which is shown in Figure 4.3. For instance, the loop "Idle, τ_1 running, Idle" needs at least C_1 units of time to be completed. It is easy to see that every loop in the automaton needs a minimum finite amount of time to be performed.

We now focus on the Server automaton of Figure 4.2. Observe that most transitions are guarded by synchronisation labels, except:

- A) from Active to Executing
- B) from Executing to Active
- C) from Empty to Idle
- D) from Executing to Recharging
- E) from Recharging to Active

All synchronisation labels are ultimately triggered by events of the tasks and the scheduler. In particular, scheduling events "active" and "empty" are finite in number over a finite interval of time. Therefore, for every run of finite duration, the number of event transitions in the run that include synchronisation labels is finite.

It remains to analyse the cycles which do not contain any synchronisation label; there are only two of those, the cycle involving transitions A), D) and E); and the cycle containing transition A) and B).

Regarding the first one, notice that clock x is reset on E), and it is checked with an equality in D); therefore, the duration of each cycle is at least Q instants, and the number of occurrences of these cycles in any finite interval of time is finite.

Regarding the second one (A, B), we can have an unbounded number of *consecutive* occurrences of the cycle in any bounded interval of time of length T. Let us fix an arbitrary large number N: then it is possible to find a run ρ of length $\mathsf{len}(\rho) > N$, by moving into location Active before T, and then looping with $N/2$ cycles, using transitions A) and B), in an arbitrary small amount of time. Let π be the timed path corresponding to ρ, and let s_n be the last state visited by ρ. By applying operator $\mathsf{Cnt}^*(\pi)$ we obtain an equivalent path, i.e. a path with the same duration of π and that reaches the same state s_n and that contains only one simple cycle between Active and Executing, and its length is equal to $\mathsf{len}(\rho) - N + 2$.

Clearly, the length of any path of bounded duration is also bounded. By applying combinatorial arguments similar to the ones used in [BDG+11] we can upper bound the length of any run with

$$\forall \rho, \mathsf{len}(\rho) \leq 2|\mathsf{Var}| + (2|\mathsf{Var}| + 1) \cdot |Loc| \cdot (2^{|Trans|+1} + 1).$$

The number of such paths is finite, so reachability is decidable. □

4.7 Evaluation

We have implemented the HSC analysis in the software tool FOrmal Real-Time Scheduler (FORTS) [SL14a]. FORTS is a model checker targeting real-time scheduling problem and it accepts LHA models as input. We use it here for reachability analysis in HSC.

4.7.1 Comparison with the Lipari-Bini test

We used the tool to compare the results of our analysis against the test proposed by Lipari and Bini in [LB04]. We modeled the same application described and analysed in [LB04]. In Table 4.1 we report the parameters of the task set, which has a total utilisation of 47%.

In this experiment, we checked the schedulability of the task set in servers with different values of (Q, P). In particular, we tested all integer values of $P \in [1, 27]$, and all values of $Q \in [1, P]$. The results

Task	O_i	C_i	D_i	T_i	p_i
τ_1	0	2	8	8	1
τ_2	0	2	20	20	2
τ_3	0	6	50	50	3

Table 4.1: Parameters of the example application.

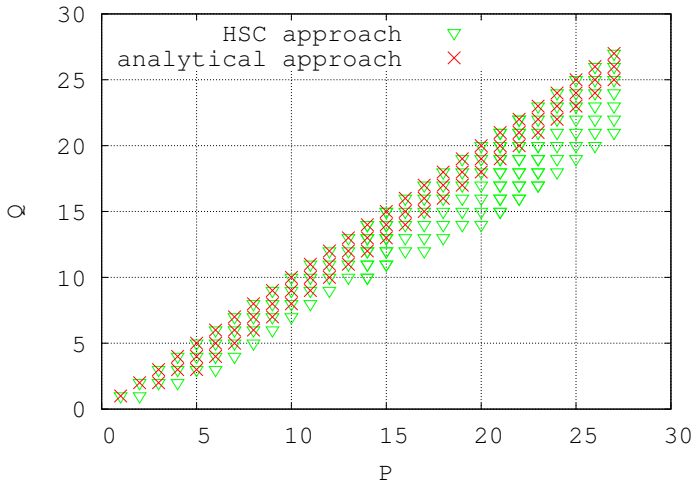


Figure 4.4: Feasible server parameters. Crosses are the schedulable pairs (Q, P) found by the Lipari-Bini test [LB04]; triangles are the ones found by our analysis.

are shown in Figure 4.4: the crosses are the results of the Lipari-Bini test, whereas the green triangles are the results of the HSC model. As you can see, the latter found many more schedulable points. In particular, point $(Q = 3, P = 6)$, which leads to an utilisation of 50% (only 3% larger than the task set utilisation) was found by the HSC, while it was not found by the analytic model.

The reason for this difference is that the Lipari-Bini test makes worst-case assumptions on the maximum delay that an application task can experience. In particular, this test assumes that, when the highest priority task is activated, it may have to wait up to $2(P - Q)$. Since the highest priority task has computation time $C_1 = 2$ and relative deadline $D_1 = 8$, it necessarily follows that $(P - Q) \leq 3$. In Figure 4.4, it is possible to note that this is always true for the Lipari-Bini test. However, the worst case initial delay may never happen: the HSC model shows that in many cases $(P - Q) > 3$, and for large Ps this can be as large as 6. How is it possible?

To understand what happens, consider the case of $P = 22$ and $Q = 16$. According to the analytic method, the worst case happens when the highest priority task arrives and, at the same instant, the server has just exhausted its budget. Apparently, this seems to be the case of time $t = 16$, when the server budget $Q = 16$ has just been exhausted. However, notice that the first busy period starting at time $t = 0$ lasts only for 12 units: therefore, at time 12 the server moves to location Empty, and from there, it moves to location Active at time $t = 16$ (due to the arrival of τ_1), where it can spend at most two units of time before moving to location Executing and completing the requested $C_1 = 2$ units of execution time. In other words, it never happens that the application can completely deplete the budget of the server. This fact is not taken into consideration by the analytic method, which then produces pessimistic results.

4.7.2 External service test

As discussed in the previous sections, the LHA formalism has the advantage of being more expressive, in the sense that it allows designers to model and analyse complex scheduling scenarios that cannot be expressed easily as a set of independent periodic real-time tasks.

To demonstrate the expressiveness of the model, let us consider an application consisting of just two real-time periodic tasks, whose parameters are reported in Table 4.2. Each task provides a service that is requested by the external environment (i.e. by other applications, or by an interrupt). Each task τ_i has an incoming queue of requests: at its periodic activation time it checks the contents of the queue: if there is no request, it executes for very little time C'_i; if there is one request, it will execute for C''_i; if there are two or more requests it will execute for C'''_i. Therefore, the actual load generated by the application depends on the number of external requests per task.

Task	C'_i	C''_i	C'''_i	D_i	T_i	p_i
τ_1	1	2	3	12	12	1
τ_2	1	3	5	15	15	2

Table 4.2: Parameters of the external service application.

In our example, we model the two request queues with simple counters w_1, w_2, both initialised to be 0. At its arrival time, each task reads its counter, sets its computation time to the corresponding value, and resets the counter to 0. In the LHA formalism, a discrete variable like w_i can be automatically encoded in the location signature.

The arrival of external requests is modeled by the Service automaton shown in Figure 4.5. Initially, the automaton waits non deterministically for an interval of time between 0 and its maximum initial offset O_r. Then, every T_r units of time, it produces one request for either τ_1 or τ_2, and the choice is again non deterministic.

It is not easy to compute the worst-case load produced by the application: if we want to use classic schedulability analysis, we need to analyse all possible combinations of requests to the two tasks. In fact, the Service automaton can request only one service at time, and depending on the values of T_r, several possible combinations of service requests may generate the worst-case load.

However, our HSC automaton does exactly this: it checks all possible combinations of service requests, and verifies if the system is schedulable under all possible cases. By setting $T_r = 10$ and $O_r = 0$ and applying the analysis for different values of the pairs (Q, P), we obtained the results shown in Figure 4.6. Notice that the worst-case utilisation of the task set, without considering the Service automaton, is $\frac{C'''_1}{T_1} + \frac{C'''_2}{T_2} = 58.333\%$. However, the minimum fraction $\frac{Q}{P}$ found by our analysis is 50%, corresponding

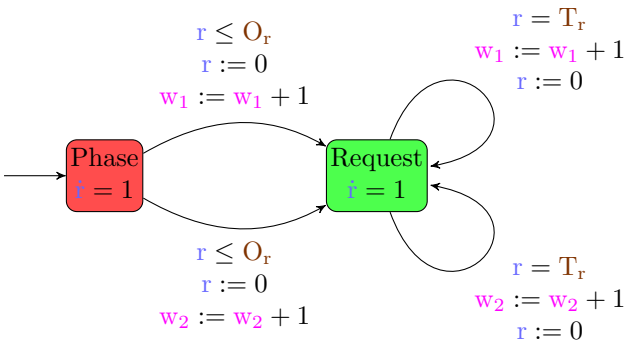


Figure 4.5: The service request automaton

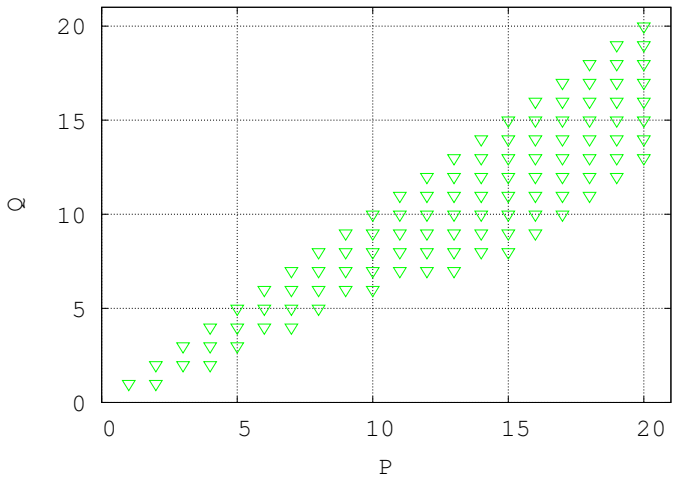


Figure 4.6: Feasible server parameters for external service test

to the two pairs $(Q = 1, P = 2)$ and $(Q = 2, P = 4)$. In fact, by analysing all possible combinations of service requests, the largest utilisation needed is indeed 50%, exactly equal to what our analysis found. This value corresponds to the case when the first task τ_1 executes for $C_1'' = 2$ (serving one request), whereas the second task τ_2 executes for $C_2''' = 5$ (serving two requests).

Also, notice that the pair $(Q = 8, P = 15)$ provides schedulability with server utilisation equal to 53.34% and a relatively large P. In general, a large P is desirable because it reduces the overhead of switching between different applications. In this case, our analysis shows that we can set a period larger than the smaller period in the application, and still achieve a relatively low resource utilisation.

4.7.3 A real case study of an avionics system

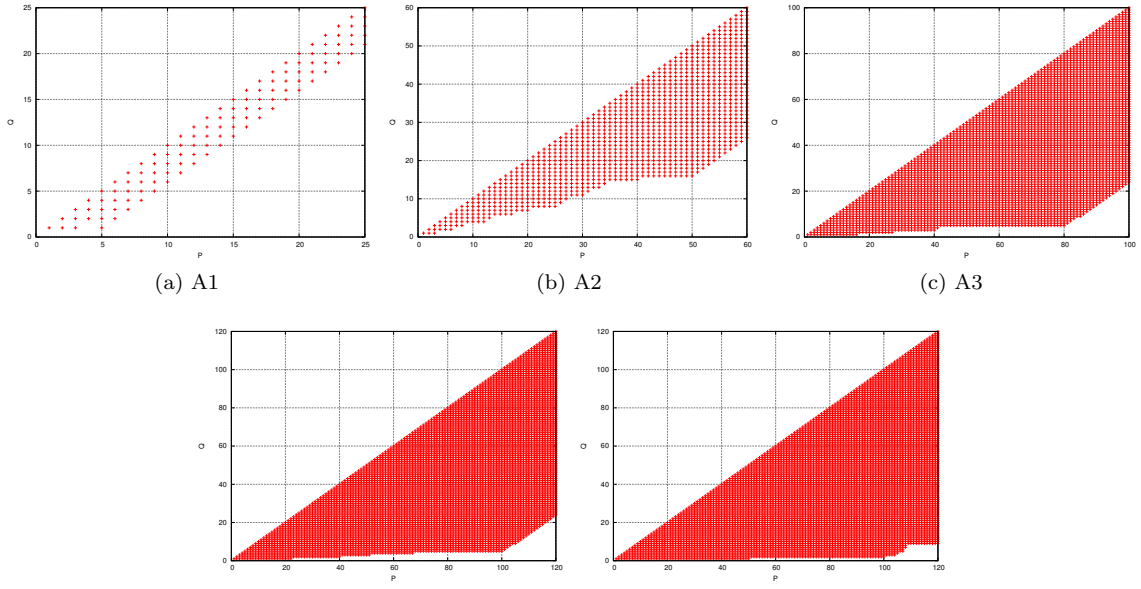
The case study we use here was originally described in [Dod06a] and [Dod06b]; it was later adapted to hierarchical scheduling in [CPV13]. It consist of fifteen real-time tasks with very different values of periods. Carnevali *et al.* [CPV13] partitioned the task set into five components and verified the schedulability of each partition under TDM with a pre-defined pattern of time slot assignment. The tasks and the components are reported in Table 4.3, where time is expressed in milliseconds.

Component	Task	O	C	T	D	p	U
$\mathfrak{A}_1$	τ_{11}	0	1	10	5	1	0.2
	τ_{12}	0	1	40	40	2	
	τ_{13}	10	2	40	40	3	
	τ_{14}	20	1	40	40	4	
$\mathfrak{A}_2$	τ_{21}	0	1	40	40	1	0.305
	τ_{22}	0	5	50	50	2	
	τ_{23}	10	4	50	50	3	
	τ_{24}	16	5	50	50	4	
$\mathfrak{A}_3$	τ_{31}	2	4	80	80	1	0.06
	τ_{32}	15	1	100	100	2	
$\mathfrak{A}_4$	τ_{41}	0	4	100	100	1	0.045
	τ_{42}	10	1	200	200	2	
$\mathfrak{A}_5$	τ_{51}	10	1	200	200	1	0.019
	τ_{52}	3	4	400	400	2	
	τ_{53}	0	4	1000	1000	3	

Table 4.3: Specification of the avionics case study

In our analysis, we used the same components as in [CPV13]. We modeled each partition using a HSC automaton. Then, we performed the analysis of the entire system in two steps: we first analysed each component individually. The profile of valid pairs (P, Q) for each component is shown in Figure 4.7, and the pairs which lead to the minimal utilisation for each component are listed in the fourth column of Table 4.4.

In the second step, we performed the "integration" by selecting the combination of pairs (P, Q) for each component so that the overall utilisation is less than 100%. Notice that, by using dynamic periodic servers, we can easily select different values for the periods and the budget of the different applications so to minimise some cost function. For example, one objective can be to maximise the values of the servers periods: in fact, small periods imply a more frequent switch between components, and hence a greater overhead. One possible choice for each parameters is reported in the fifth and last column of Table 4.4.



(a) A1 (b) A2 (c) A3

(d) A4 (e) A5

Figure 4.7: Feasible (P, Q) space for each component of the avionics case study.

	Orig. Util.	Min Utilisation		Reduced Overhead	
$\mathfrak{A}_1$	0.2	(5,1)	$Q/P = 0.2$	(5,1)	$Q/P = 0.2$
$\mathfrak{A}_2$	0.305	(50,16)	$Q/P = 0.32$	(50,16)	$Q/P = 0.32$
$\mathfrak{A}_3$	0.06	(80,5)	$Q/P = 0.0625$	(100,24)	$Q/P = 0.24$
$\mathfrak{A}_4$	0.045	(22,1)	$Q/P = 0.0455$	(100,5)	$Q/P = 0.05$
$\mathfrak{A}_5$	0.019	(100,2)	$Q/P = 0.02$	(200,9)	$Q/P = 0.045$
U_{tot}	0.629		0.648		0.855

Table 4.4: Server parameters for the avionics case study.

4.7.4 Scalability of the analysis

A full-fledged analysis of the run-time complexity of our model is out of the scope of this thesis. Nevertheless, it is important to briefly discuss the scalability of our analysis with respect to the size of the model. First of all, a few *caveat*. It is well-known that formal methods suffer from the so called *state-space explosion* problem: the number of states to analyse is exponential in the size of the input.

Therefore, an exponential dependency from the number of tasks in the application is unavoidable. The important issue here is to understand to which extent the analysis is still doable with modern computer systems.

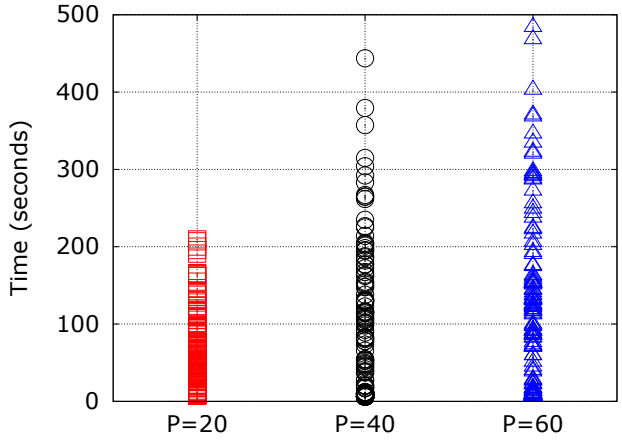


Figure 4.8: Run-time of FORTS on a 8 task model

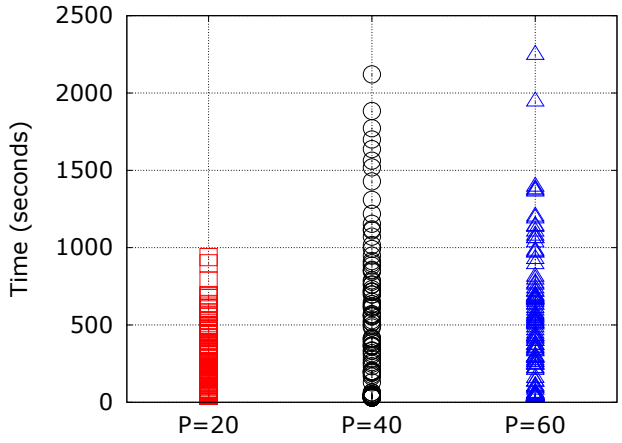


Figure 4.9: Run-time of FORTS on a 10 task model

N	8			10		
P	20	40	60	20	40	60
max time (s)	209	443	483	938	2120	2244
ave time (s)	63	85	104	245	371	384
max memory (M)	79	145	178	229	444	527
ave memory (M)	26	35	44	69	107	126

Table 4.5: Run-time results of HSC

The experiment was conducted on a common MacBook with Intel(R) Core(TM) i5 CPU @ 2.5GHz

and 8 GB of RAM. We run tests for different task sets and different values of the parameters. Each test is specified by a tuple (N, U, P) , where $N \in \{8, 10\}$ is the task set size, U is task set utilisation, and $P \in \{20, 40, 60\}$ is the period of a server. We also fix $\frac{Q}{P} = 0.6$. For each pair (N, P) 150 task sets would be randomly generated according to the Randfixedsum algorithm [ESD10]. For a task set, its utilization U is randomly sampled in the range $[0.2, 0.6]$. Task periods are selected in the range $[10, 100]$ using log-uniform sampling and the minimum granularity of periods is 10. Each task's initial phase is uniformly distributed between 0 and its period. We know that the complexity of reachability analysis in a HSC is directly related to the hyperperiod length of tasks and server. We constrain this hyperperiod length to be less than 2000. Task priorities are assigned by Rate Monotonic scheduling: a task with shorter period will be given higher priority; the priority relation between two tasks with the same period is randomly selected. Furthermore, we avoid the generation of task τ_i, with $D_i - C_i < P - Q$, which will trivially miss their deadline.

For each task set, we measured the time (in seconds) and memory (in MB) needed to decide schedulability. The run-time results are reported in Figure 4.8 and Figure 4.9; more detailed statistical results are in Table 4.5. A first observation is that larger P will result in higher cost of HSC analysis, due to the non-determinism in the Server automaton. And we get the worst-case scenario when $P = 60$ and $N = 10$, under which circumstance the longest running time for one HSC is around 37 minutes, whereas the memory cost is always less than 530 MB. We believe that, by introducing parallelism in the analysis tool and by carefully optimising the code we can achieve higher performance.

4.8 Conclusion

We presented a formal model of a dynamic server algorithm for hierarchical scheduling that can be used for component-based analysis of hierarchical real-time systems. The model is based on the very expressive formalism of Linear Hybrid Automata. We have shown that the model provides more precise results than classic analytic schedulability formulas, and allows to model components with complex dependencies. We have run extensive simulations to demonstrate that the model can be analysed efficiently for components with 10 real-time periodic tasks.

The proposed model is very general but it does not account yet for the overhead of context switch between components. Also, the impact of memory access and caches on the execution time of the tasks has been neglected. We are currently working on a more accurate model that can account for the scheduling overhead and the cache-related preemption delay caused by other components in the system.

Chapter 5

Exact G-FP Schedulability Analysis

In this chapter we present an exact schedulability test for sporadic real-time tasks scheduled by the Global Fixed Priority (G-FP) Fully Preemptive Scheduler on a multiprocessor system. The analysis consists in modeling the system as a Linear Hybrid Automaton (LHA), and in performing a reachability analysis for states representing deadline miss conditions. To mitigate the problem of state space explosion, we propose a pre-order relationship over the symbolic states of the model: states that are simulated by others can be safely eliminated from the state space.

We also formulate the concept of decidability interval with respect to a set of constrained-deadline sporadic tasks on multiprocessors. The decidability interval is a bounded time interval such that, if a deadline miss occurs in the schedule, then it is possible to find a configuration of arrival times for the tasks such that the deadline miss happens within the bounded interval. Vice versa, if no configuration of arrival times produces a deadline miss in the bounded interval, then no deadline miss is ever possible in the schedule. Hence we prove that the schedulability analysis problem is decidable, and we provide a formula for computing the decidability interval. To our knowledge, this is the first time such a time interval is proposed for sporadic tasks running on multiprocessors.

The proposed schedulability analysis has been implemented in our software tool FOrmal Real-Time Scheduler (FORTS) [SL14a]. For the first time we assess the pessimism of the state-of-the-art approximate schedulability test through experiments. Moreover, we show that the use of the proposed model permits to analyse tasks with more general parameter values than other exact algorithms in the literature. Nevertheless, even with our approach the complexity remains too high for analysing practical task sets with more than 7 tasks.

The content in this chapter is based on [SL14b] and [SLa].

5.1 Introduction

Since the seminal work of [LL73], the fixed-priority scheduling problem has been extensively studied. The problem has been solved exactly for single processor systems by using a well known property: the worst-case response time of a task happens when it is activated simultaneously with its higher priority tasks, and all jobs are activated at their maximum frequency. Therefore, it suffices to simulate the system starting from this *critical instant* and activating all subsequent jobs as soon as possible, until the first idle time.

We consider the problem of checking the schedulability of a set of independent real-time sporadic tasks on a multiprocessor platform when the scheduling algorithm is the *Global Fixed Priority* (G-FP) Fully Preemptive scheduler. According to this scheduling algorithm, on a m-processor platform all jobs

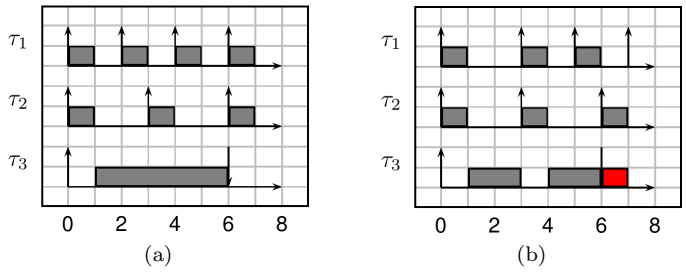


Figure 5.1: Example of schedule of sporadic tasks (a) jobs arrive as soon as possible (b) second job of τ_1 is delayed.

are ordered in one single ready queue by decreasing priority, and the m highest priority jobs are executed at every instant.

Unfortunately, there is no easy solution for checking the schedulability of a task set scheduled by G-FP. The difficulty comes from two facts.

- No single *critical instant* exists: the worst-case response time of a task can be found anywhere in the schedule. Also, it is not true that the worst-case response time happens when all jobs are activated as soon as possible. An example is presented in the following.
- On the other hand, the sporadic behaviour of the tasks increases the number of possible interleavings.

To better understand the problem, consider the following example (from [Bar07]): the system consists of 3 tasks $\tau_1 = (1, 1, 2)$, $\tau_2 = (1, 3, 3)$ and $\tau_3 = (5, 6, 6)$, to be scheduled by G-FP on a 2-processor platform. Task τ_1 has the highest priority and τ_3 is the lowest priority task. The schedule obtained when all tasks start at time 0 and arrive as soon as possible is shown in Figure 5.1a, where all tasks meet their deadlines. However, if the second job of task τ_1 arrives at time instant 3 instead of 2, task τ_3 misses its deadline (Figure 5.1b).

In fact, we cannot make any worst-case assumption on the arrival times of the jobs. In order to find the exact combination of arrival times that leads to the worst-case response time of a task, it is then necessary to explore all possible legal combinations of arrivals, and this number is so large that a brute-force approach fails already for very small task sets.

Therefore, most of the research in the literature has been focused on finding upper bounds to the response times (as we are going to see in Chapter 6). However, to assess the pessimism of such approximate analyses, it is necessary to solve the problem exactly, i.e. to obtain necessary and sufficient conditions for the schedulability of a task set.

Contributions In this part, we address the problem of deriving an *exact analysis* for the schedulability of a set of sporadic real-time tasks scheduled by G-FP on a multiprocessor platform. We model the problem using the formalism of Linear Hybrid Automata to represent the tasks and the scheduler. In particular, deadline miss conditions are modeled as error locations in the automata. The analysis consists in performing a reachability analysis for such error states. Due to the non-deterministic sporadic task activations, the analysis complexity explodes for very small task sets. To defer the state explosion, we propose a *weak simulation relation* between symbolic states and prove its correctness. The relation allows us to eliminate those states that are not useful for our reachability analysis, thus reducing the size of the state space. Furthermore, we prove the decidability of the proposed analysis by demonstrating that the schedulability test of a set of sporadic tasks under G-FP scheduling policy can be done in a bounded time interval, called *decidability interval*. We present the implementation of our model in a software tool, and we show that it can handle more complex task sets with respect to state-of-the-art

exact algorithms based on discrete time. Here, by "more complex" we mean that tasks' parameters can be generic values and, as we are going to see, this makes a critical difference between previous work on exact multiprocessor schedulability analysis and our solution. Also, we evaluate the pessimism of current state-of-the-art approximate schedulability analysis of G-FP scheduling over sporadic tasks. Through extensive experiments, we investigate the factors that can affect the run-time performance of the proposed schedulability analysis.

Limitations Unfortunately, as the number of sporadic tasks grows beyond 7, and for more than 4 processors, the complexity rises so much that all exact analysis techniques proposed so far can hardly terminate on current desktop computers, even when using our weak simulation relation. This is due to the exponential nature of the problem and it can only be mitigated by future improvements of the simulation relation. Thus, we will continue to work in this direction in the hope to further enhance the practicability of the method.

5.2 Related Work

The general properties of multiprocessor scheduling have been discussed in many previous works. [CG07] and [GGCG13] proposed upper bounds to the *feasibility interval* of a set of *periodic* tasks scheduled upon a multiprocessor (uniform or heterogeneous).

Regarding exact analysis of sporadic tasks, the first brute force approach to the problem was proposed in [BC07a]: the test assumes discrete time parameters, and it consists in building a finite state machine that represents all possible combinations of arrival times and execution sequences for a task set scheduled by G-EDF. Unfortunately, the problem is so complex that the authors can analyse only tasks whose period is in the range $\{3, 4, 5\}$; the tool produces an out-of-memory error for values of $T = 6$.

[CGG11] proposed an exact schedulability test for a set of *periodic tasks*, but they did provide neither a tool, nor experiments with task sets. We believe that their algorithm is very complex and a naive implementation would not scale to a large number of tasks. [GGD+07] proposed a Timed Automata model for schedulability analysis of *periodic tasks*. However, periodic tasks are simpler to analyse than sporadic tasks: we will provide a detailed comparison in Section 5.6.4.

Recently, [GGL13] improved over [BC07a] by using an *antichain* technique. In particular, they proposed a *simulation relation* between states of the underlying finite automaton. An informal definition of simulation relation is the following:

> Given two states s_1 and s_2, we say that s_1 simulates s_2 (denoted as $s_1 \succeq s_2$) if and only if: 1) for every state s_2' successor of s_2, there exists a state s_1' successor of s_1 and $s_1' \succeq s_2'$; 2) if s_2 is an error state (i.e. it models a deadline miss), then also s_1 is an error state.

Thanks to this relation, when we find two states such that $s_1 \succeq s_2$, we can avoid analysing all paths starting from state s_2: in fact, if the error state is not reachable from s_1, then it is not reachable from s_2 either. This produces a significant reduction on the number of states to be analysed in the reachability analysis. The simulation relation proposed in [GGL13] is valid for any fixed job-level scheduling algorithm, including G-FP and G-EDF. Besides, [BMS12] studied the feasibility problem of sporadic tasks upon the multiprocessor by reducing it to a safety game, where the two players are the scheduler and the set of the tasks respectively. As shown in [GGS14], the antichain technique in [GGL13] can be applied to [BMS12] in order to improve the efficiency.

However, all such methods rely on explicit (discrete) techniques for time analysis and are limited to tasks with very small discrete parameters. For example, in their experiments [GGL13] can analyse task sets with maximum period equal to $T = 8$ on 2 processors.

In this chapter we take a different approach. We model the system as a Linear Hybrid Automaton (LHA) and then we perform our analysis on the corresponding symbolic state space. As in [GGL13], we define a weak simulation relation over the symbolic states, and prove its correctness for G-FP scheduling. This allows us to considerably reduce the analysis time, and thus to analyse more complex task sets. Due to the different features between explicit and symbolic techniques for state space exploration, when tasks are with small parameters, it is possible that existing works on exact multiprocessor schedulability analysis are more efficient than our solution; however, our exact analysis is the only work that can handle task sets with general configurations. Furthermore, as it will be shown in Chapter 9, the exact G-FP schedulability analysis in discrete time domain is problematic.

5.3 Multiprocessor Schedulability in LHA

In this section we describe the automaton used for modeling our scheduling problem. In particular, we use two different types of automata that synchronise with each other: the task automata and the scheduler automaton. Indeed, the LHA model we are going to propose can be also encoded by using Timed Automata with Stopwatches. As it will be clear in the next section, the only difference is that we allow some variables to decrease at unitary rate, whereas in Stopwatch Timed Automata all variables are either stopped or increasing at unitary rate. We think that using the LHA model is more straightforward to understand our analysis scheme in Section 5.4.

5.3.1 The task automata

We start by presenting the LHA that models one single sporadic task. Such a LHA model for the sporadic task is called *task automaton*. A concrete task automaton $\mathsf{TA} = (\mathrm{C}, \mathrm{D}, \mathrm{T})$ is depicted in Figure 5.2. It has two continuous variables p and c, and four locations.

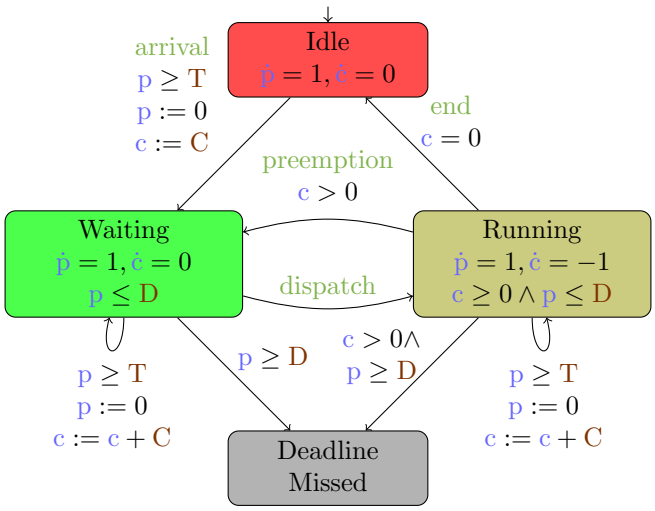


Figure 5.2: Task Automaton

Variable p represents the *time passed* since the latest activation of the task, and its rate is always 1. Every time a new job arrives, p is reset to 0. Variable c represents the *remaining computation time* of a task. Its rate can be 0, when the task does not execute, or -1 when the task executes.

The automaton works as follows. Initially, it is in state Idle, where $p \geq 0$ and $c = 0$; $p \geq 0$ models the fact that the first job release of a task can happen at any time. From there, when the guard constraint $p \geq T$ is satisfied, i.e. at least T time units have been passed since latest activation of the task, it can

move non-deterministically to location Waiting. Along with this new job arrival transition, p variable is reset to 0 and C is assigned to variable c. Also, it synchronises with the scheduler (see next section) on the task arrival label. Notice that every task always executes for its WCET: G-FP is *sustainable* [BB09b] and if the system is schedulable when every task always executes for its WCET, it is also schedulable when a task is allowed to execute for less than its WCET.

While in Waiting, the rate of c will remain equal to 0. The automaton moves to location Running after synchronising with the scheduler on label dispatch, which notifies that some processor is available for the task to run. While a task is running, the rate of c is set to -1, so its remaining computation time decreases. Its execution can be preempted by the arrival of a higher priority task, at which point the task will move back to location Waiting after synchronising with the scheduler on label preemption.

We say a task is *active* if it is in location Waiting or Running. An active task must finish its computation time before reaching its deadline. This means the c variable must reach 0 no later than the time at which p reaches D. Otherwise, a task misses its deadline and goes (from Waiting or Running) to the DeadlineMissed location. If a task finishes its execution before deadline, i.e. $c = 0$ and $p \leq D$, the task is forced to move to location Idle (transition from Running to Idle).

In case that a task has unconstrained deadline, there can be a new job arrival for an active task. This is modeled as a non-deterministic transition from Waiting or Running to itself. Since the new instance must wait for its precedence completes, variable c is incremented by C with the transition.

In the following we will denote as $\mathsf{TA_i}$ the automaton corresponding to the ith task in the system, with q_i, c_i, p_i its location, left computation time and passed time, respectively, and with arrival_i, end_i, dispatch_i, preemption_i the corresponding synchronisation labels.

5.3.2 Scheduling automaton

Given a set of tasks $\mathcal{T} = \{\mathsf{TA_1}, \ldots, \mathsf{TA_n}\}$, set A is defined as the set of active tasks that are in locations Waiting or Running, and set R denotes the set of tasks that are in location Running.

Let $\mathsf{Scheduler} : 2^{\mathcal{T}} \to 2^{\mathcal{T}}$ be a *scheduling function* that, given a set of active tasks, returns the set of executing tasks: $\mathsf{R} = \mathsf{Scheduler}(\mathsf{A})$.

We consider a G-FP Scheduler, which chooses the $\min\{m, |\mathsf{A}|\}$ highest priority tasks to run.

The scheduling function can be modeled by a finite automaton synchronised with the task automata the system is composed of. More formally, the scheduling (or scheduler) automaton $\mathsf{Sched} = \{m, \mathsf{Loc}, \mathsf{Lab}\}$ is characterised by:

- m is the number of identical processors in the system;
- Loc is the set of locations of the scheduler;
- $\mathsf{Lab} = \bigcup_i Lab_i$ with $Lab_i = \{\text{arrival}_i, \text{end}_i, \text{dispatch}_i, \text{preemption}_i\}$ is the set of synchronisation labels.

The responsibility of a scheduling automaton is to synchronise with the task automata, i.e. to decide which tasks to run (staying in location Running) and which tasks to wait (staying in location Waiting). Every time a task completes its execution or releases a new job, the active task set A changes to A' and a new running task set R' is computed according to the scheduling function $\mathsf{R}' = \mathsf{Scheduler}(\mathsf{A}')$. Then, for the task that is in R but was not in R', the scheduling automaton informs its preemption from the processor through synchronisation on the preemption_i label; and for the task that was not in R but is now in R', the scheduling automaton synchronises on the dispatch_i label with it.

An example of scheduling automaton for $n = 3$ tasks on $m = 2$ processors is shown in Figure 5.3. In the figure, nodes depict locations, and the name of the location encodes the state of the system queue,

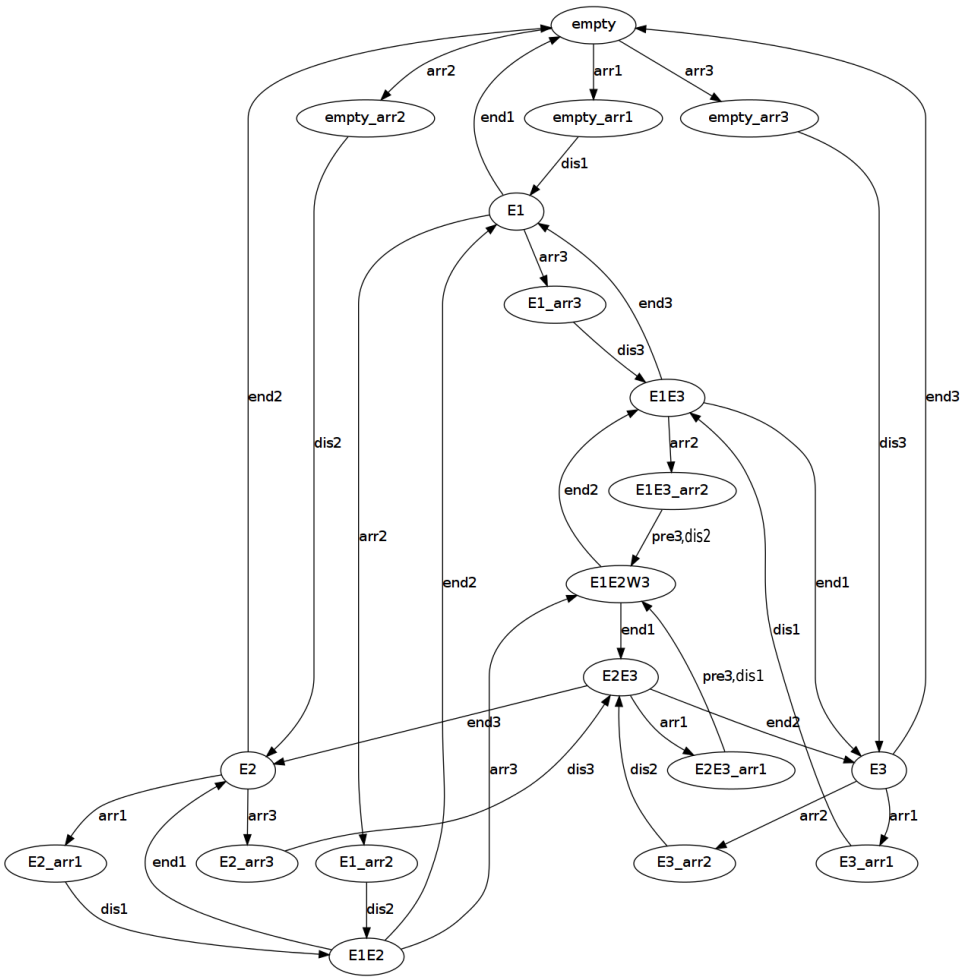


Figure 5.3: Scheduler for 3 tasks on 2 processors

and in some cases the event that just happened. For example, location E1E2W3 corresponds to the execution of task τ_1 and τ_2 on the two processors, and the task τ_3 waiting to be executed; location E1_arr2 represents the fact that, while task τ_1 is executing on one processor, task τ_2 has just arrived. Also, please note that all locations with names containing arr are assumed to be *committed locations*. Finally, on the edges we show the synchronisation labels (in short form for graphical reasons), hence arr1 stands for arrival_1, etc. For simplicity, when there is a preemption (e.g. τ_3 is preempted by the arrival of τ_1), we put the two synchronisation labels (e.g. pre3 and dis1) on the same transition. This means between the two synchronisations, no time will elapse (as we assume no context switch cost). This can be realised by inserting a committed location in between the two. In the special case of one processor, a formal modeling of the fixed-priority scheduler can be found in Chapter 3

The number of locations needed for representing the scheduler automaton is exponential in the number of tasks. Such locations can be automatically generated by using function Scheduler() for computing which task to execute and which task to suspend or preempt. Notice also that the location encodes the same information that is contained in the task automata presented above; in particular, executing tasks will be in location Running, whereas suspended tasks will be in location Waiting. Therefore, the scheduler automaton does not add additional complexity to the problem; on the contrary, it restricts the number of possible combinations of task locations: for example a lower priority task cannot be in the Running location if there are m higher priority tasks that are active.

Finally, a system automaton $\mathsf{SA} = (\mathcal{T}, \mathsf{Sched})$, is the parallel composition of n task automata and

one scheduler automaton, where

- $\mathcal{T} = \{\mathsf{TA}_1, \ldots, \mathsf{TA}_n\}$ is a set of n task automata;
- Sched is the scheduler automaton.

The following theorem shows that SA models the real-time scheduling of a set $\mathcal{T}$ of sporadic tasks with G-FP, including all legal task arrival and execution patterns.

Theorem 5. *Given a task set $\mathcal{T}$, all tasks in $\mathcal{T}$ are guaranteed to meet their deadlines if and only if* DeadlineMissed *locations are not reachable in the system automaton* SA.

Proof. We must demonstrate that, if there is a deadline miss for the set of tasks under G-FP scheduling, there is a sequence of transitions in SA starting from the initial location until to the DeadlineMissed location. Vice versa, if some DeadlineMissed locations can be reached using a sequence of transitions, then there exists a configuration of arrival times for the sporadic instances such that one task will miss its deadline.

Suppose that the task set is not schedulable under G-FP and the first deadline miss happens at time t. We now build a sequence of transitions (i.e. a *trace*) in the SA that reproduces the schedule.

Suppose that the first instance of a task τ_i is released at time $r_{i,1}$. Then, the corresponding TA remains in the Idle location while $\mathrm{p} < r_{i,1}$, and the transition to location Waiting is taken when $\mathrm{p} = r_{i,1}$; correspondingly p is reset to 0. The scheduling automaton Sched at each instant represents the state of the ready queue, so it mimics the state of the schedule. For example, upon arrival of a high priority task, a sequence of transitions synchronized by dispatch and/or preemption is triggered in zero time so that the TA corresponding to the high priority task is moved to location Running, whereas the TA corresponding to the preempted task is moved to location Waiting. If a task executes for Δt units of time, the corresponding variable c is decreased by the same amount, so the value of the variable always represents the execution time.

Successive releases at $r_{i,j}$ are treated differently: if the task has already completed its execution, the corresponding automaton is in location Idle, so they are treated as in the first release.

If at time $r_{i,j}$ the previous instance of τ_i has not yet completed, the TA may decide non-deterministically to update c and p to account for the new instance or decide to ignore the new arrival and analyse the previous one. In practice, in the analysis we need to consider both cases. If the instance that misses the deadline is the $(j-1)$-th instance of task τ_i, we ignore the release of the $j-th$ instance (which has no impact on the current schedule), and continue the analysis of the current instance until the DeadlineMissed location is reached.

If instead the deadline miss happens on a different instance of the same task or of another tasks, we update variable $\mathrm{c} \leftarrow \mathrm{c} + \mathrm{C}$ and variable $\mathrm{p} \leftarrow 0$ to account for the new instance.

Then, it is clear that each task arrival, execution, preemption and completion corresponds to a transition in the task automaton TA. Therefore, a deadline miss for a task always corresponds to the corresponding TA going into the DeadlineMissed location.

Vice versa, if there exists a sequence of transitions in SA that finally reaches the DeadlineMissed location, following the same reasoning as above, we can find a corresponding task arrival and preemption pattern in the scheduling of tasks that causes a deadline miss.

In conclusion, the task set is schedulable if and only if the location DeadlineMissed is not reachable in the system automaton SA. □

Given a state s in SA, $\mathsf{A}(s)$ and $\mathsf{R}(s)$ represent the set of active and running tasks in the system respectively. We denote with q_0 the current location of the scheduler automaton, and with q_i $(1 \leq i \leq n)$ the location of the ith task automaton. The same notions are also applied to a symbolic state S.

5.4 Weak Simulation Relation in SA

In the previous section we proved that analysing the schedulability of a task set is equivalent to performing a reachability analysis of DeadlineMissed locations in SA. Due to the non-deterministic and sporadic behaviour of task arrivals, the exploration of SA's (symbolic) state space will easily produce a "state explosion". To reduce the number of generated states, we propose a weak simulation relation for SA such that, given two states S_1 and S_2, if S_1 simulates S_2 then S_2 can be eliminated from state space without interfering with the final schedulability analysis result.

5.4.1 Weak simulation in concrete state space

We first discuss the weak simulation relation in concrete state space of SA, then we will extend it for symbolic states.

Definition 5. *A weak simulation relation in the concrete state space of* SA *is a pre-order* $\succeq \subseteq \mathsf{space} \times \mathsf{space}$ *such that the following two conditions are satisfied:*

1. $\forall s_1, s_2, s_4$ *s.t.* $s_1 \succeq s_2, s_2 \rightarrow s_4$*: there exists* s_3 *s.t.* $s_1 \Rightarrow s_3$ *and* $s_3 \succeq s_4$*;*
2. $\forall s_1, s_2$ *s.t.* $s_1 \succeq s_2 : \forall i$ $s_2.q_i = \text{DeadlineMissed}$ *implies* $s_1.q_i = \text{DeadlineMissed}$*.*

If $s_1 \succeq s_2$*, we say that* s_1 *simulates* s_2*. If* $s_1 \succeq s_2$ *but* $s_2 \not\succeq s_1$*, we write* $s_1 \succ s_2$*.*

Roughly speaking, $s_1 \succeq s_2$ means that the scenario in s_1 is worse than in s_2 for tasks to finish execution before their deadlines.

The first condition in Definition 5 says that, given $s_1 \succeq s_2$ and given s_1 that performs a (discrete or time) step to s_4, there exists a state s_3 reachable from s_1 such that $s_3 \succeq s_4$. The second condition says that if some task in the simulated state (s_2) misses its deadline, so does it in the simulating state (s_1). For these two reasons, during state space exploration, we can safely eliminate states that are simulated by others without violating the reachability analysis result. Note that for two states $s_1 \succeq s_2$ and $s_2 \succeq s_1$, we only need to keep one of them.

Now, we present a specific pre-order relation in space(SA) that satisfies the definition of a weak simulation relation, thus it can be used to simplify the state space exploration in SA.

Definition 6 (Slack-time pre-order)**.** *For the* SA *automaton, the* slack-time *pre-order* $\succeq_{st} \subseteq \mathsf{space} \times \mathsf{space}$ *is defined as follows:* $\forall s_1, s_2$*,* $s_1 \succeq_{st} s_2$ *if and only if*

$$\forall \tau_i : \quad s_1.p_i \geq s_2.p_i \quad \wedge \quad s_1.c_i \geq s_2.c_i$$

For an active task, the difference between the time left before its deadline and the remaining computation time is called the *slack* of the task. When the slack is less than 0, the task is doomed to miss its deadline. Intuitively, a smaller slack corresponds to a more urgent scenario. In a task automaton $\mathsf{TA_i}$, the slack can be computed by $D_i - p_i - c_i$. Given two states s_1, s_2 such that $s_1 \succeq_{st} s_2$, there is $s_1.p_i \geq s_2.p_i$ and $s_1.c_i \geq s_2.c_i$. So, an active task's slack in s_1 is no larger than its slack (if in s_2 it is also active) in s_2.

We now prove that $\succeq_{st}$ satisfies the two conditions for a weak simulation relation in Definition 5.

Theorem 6. *The pre-order relation* $\succeq_{st}$ *is a weak simulation relation in* space(SA)*.*

Proof. To prove that $\succeq_{st}$ is indeed a weak simulation relation, we must demonstrate that it satisfies the two properties stated in Definition 5, where the second point trivially holds for $\succeq_{st}$. Therefore, in this

proof we address the first point: i.e. given $s_1 \succeq_{st} s_2$ and $s_2 \rightarrow s_4$, we prove that there exists s_3 such that $s_1 \Rightarrow s_3$ and $s_3 \succeq_{st} s_4$.

$s_2 \rightarrow s_4$ in SA can be a time step or a discrete step. The latter can be further differentiated, depending on whether it is caused by a task arrival or by a task completion. In the following we will analyse these cases one by one.

1. *$s_2 \rightarrow s_4$ is a time step with elapsed time t:* $s_2 \xrightarrow{t} s_4$. Let us consider a timed step sequence $s_1 \overset{t}{\Rightarrow} s_3$ with the same t as accumulated time; and let us assume that there is no new task arrival during this time interval. For any task τ_i, $s_3.p_i = s_1.p_i + t \geq s_2.p_i + t = s_4.p_i$. If a task τ_i is not in $\mathsf{A}(s_2)$, then $s_2.c_i = s_4.c_i = 0$; certainly, there will be $s_3.c_i \geq s_4.c_i$. Otherwise, a key observation for the proof is that $\forall i, s_1.c_i \geq s_2.c_i$ implies $\mathsf{A}(s_2) \subseteq \mathsf{A}(s_1)$; so task τ_i in $\mathsf{A}(s_2)$ also belongs to $\mathsf{A}(s_1)$. Suppose from s_1 to s_3 (s_2 to s_4), the time that τ_i stays in location $\mathsf{Running}$ is t_1 (t_2). Since the scheduler chooses tasks to run according to their fixed priority and $\mathsf{A}(s_2) \subseteq \mathsf{A}(s_1)$, t_1 will be no larger than t_2 and $s_3.c_i = s_1.c_i - t_1 \geq s_2.c_i - t_2 = s_4.c_i$. So, we proved that $s_3 \succeq_{st} s_4$.

2. *$s_2 \rightarrow s_4$ is a discrete step caused by the arrival of a task τ_i.* For such a step, only variables of the arriving task will change. Because that $s_1.p_i \geq s_2.p_i$, there exists also a discrete step from s_1 to s_3 triggered by τ_i's new arrival job. We have $s_3.p_i = s_4.p_i = 0$ and $s_3.c_i = s_1.c_i + C_i \geq s_2.c_i + C_i = s_4.c_i$. So, we proved $s_3 \succeq_{st} s_4$.

3. *$s_2 \rightarrow s_4$ is a discrete step caused by the completion of a task τ_i.* For such a step, only variables of the finishing task will change: $s_4.p_i = s_2.p_i \leq s_1.p_i$ and $s_4.c_i = 0 \leq s_1.c_i$. Remember that in the definition of LHA, there is always a stutter transition from a location to itself. So, there is $s_1 \rightarrow s_1$ and $s_1 \succeq_{st} s_4$.

In conclusion, the pre-order $\succeq_{st}$ satisfies point one in Definition 5 also. Thus $\succeq_{st}$ is a weak simulation relation in SA. □

5.4.2 Weak simulation in symbolic state space

As continuous variables in LHA vary in dense time domain, the weak simulation relation in concrete state space cannot be applied to reachability analysis directly. In this section, we extend the slack time weak simulation relation $\succeq_{st}$ to symbolic states.

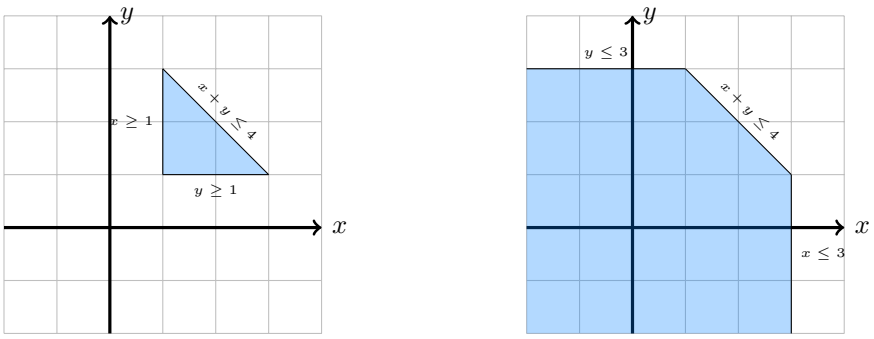
We remind here the concepts in Section 2.2 that a symbolic state is defined as a pair $S = (l, C)$, where l is a location and C is a linear constraint (or a convex region). A symbolic state abstracts a (possibly infinite) set of concrete states. We can define the weak simulation relation in symbolic state space (Space) by employing its counterpart in concrete state space (space).

Given symbolic states S_1 and S_2, we say S_1 simulates S_2 if

$$\forall s_2 \in S_2 \ , \ \exists s_1 \in S_1 \quad s.t. \quad s_1 \succeq s_2 \tag{5.1}$$

Remember that a symbolic state is a pair $(l, \mathcal{C})$ with a location l and a linear constraint $\mathcal{C}$. The linear constraint $\mathcal{C}$ can be represented by a convex region. In the following we use $\mathcal{C}$ to denote both a linear constraint and its convex region. In the context of $\succeq_{st}$ for concrete state space, there is no need to consider location names. Clearly, given two states $S_1 = (l_1, \mathcal{C}_1)$ and $S_2 = (l_2, \mathcal{C}_2)$, if $\mathcal{C}_1$ includes $\mathcal{C}_2$ (denoted as $\mathcal{C}_1 \supseteq \mathcal{C}_2$), then S_1 simulates S_2. In the following, we are going to explore a more general relationship between convex regions that can be used to judge the simulation relation between symbolic states.

Assume we are in a N-dimensional space. Given two valuations $\nu = (x_1, x_2, \ldots, x_N)$ and $\nu' = (y_1, y_2, \ldots, y_N)$, we say ν *prevails* ν', denoted by $\nu \geq \nu'$, if for all i it holds $x_i \geq y_i$. We say a valuation



(a) the original convex region (b) after windening

Figure 5.4: A convex region $\mathcal{C}$ and its windening $\nabla(\mathcal{C})$

ν is prevailed by a convex region $\mathcal{C}$ if there exists some valuation $\nu' \models \mathcal{C}$ and $\nu' \geq \nu$. Given two convex regions $\mathcal{C}_1$ and $\mathcal{C}_2$, $\mathcal{C}_1$ is said to prevail $\mathcal{C}_2$, denoted as $\mathcal{C}_1 \geq \mathcal{C}_2$ if for all $\nu \models \mathcal{C}_2$, ν is prevailed by $\mathcal{C}_1$. We can see that the prevailing relation is transitive, and convex region inclusion is a sufficient (but not necessary) condition for prevailing relation.

Given two valuations ν and ν' such that ν prevails ν', if we pair them with location names we can obtain two concrete states $s = (l, \nu)$ and $s' = (l', \nu')$. The prevailing relation between ν and ν' implies that $s \succeq_{st} s'$. Similarly, the weak simulation relation between symbolic states can be decided by employing the prevailing relation between two convex regions.

We first extend the slack time pre-order from concrete state space to symbolic state space.

Definition 7. *For the* SA *automaton, the* slack-time *pre-order* $\succeq_{st} \subseteq \mathsf{Space} \times \mathsf{Space}$ *is defined such that* $\forall S_1, S_2$, $S_1 \succeq_{st} S_2$ *if and only if* $S_1.\mathcal{C}$ *prevails* $S_2.\mathcal{C}$.

Theorem 7. *The pre-order* $\succeq_{st} \subseteq \mathsf{Space} \times \mathsf{Space}$ *is a weak simulation relation.*

Proof. From the definition of convex region prevailing. □

We now need an efficient method for checking if two convex regions are in a relationship of prevailing. To do this, we first define a widening operator ∇.

Given a convex region $\mathcal{C}$, its widening $\nabla(\mathcal{C})$ is the convex region that can be obtained as follows:

- Construct linear constraints $\mathcal{C}'$ in $2 \times N$ dimensional space $(x_1, \ldots, x_N, y_1, \ldots, y_N)$ such that

$$(y_1, \ldots, y_N) \models \mathcal{C} \quad \wedge \quad \forall i,\ x_i \leq y_i$$

- Remove the space dimensions higher than N in $\mathcal{C}'$.

$\nabla(\mathcal{C})$ represents the largest region that is prevailed by $\mathcal{C}$. $\forall \nu \in \nabla(\mathcal{C})$, there exists a $\nu' \in \mathcal{C}$ such that $\nu' \geq \nu$ and vice versa; this means $\mathcal{C} \geq \nabla(\mathcal{C})$ and $\nabla(\mathcal{C}) \geq \mathcal{C}$. An example for the widening operation is shown in Figure 5.4.

Finally, the prevailing relation between two convex regions, thus the simulation relation between two symbolic states, can be decided by the following lemma.

Lemma 2. *Given two convex regions* $\mathcal{C}_1$ *and* $\mathcal{C}_2$, $\mathcal{C}_1 \geq \mathcal{C}_2$ *if and only if* $\nabla(\mathcal{C}_1)$ *includes* $\nabla(\mathcal{C}_2)$.

Proof. We first prove that $\mathcal{C}_1 \geq \mathcal{C}_2 \Rightarrow \nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2)$. Since $\mathcal{C}_1 \geq \mathcal{C}_2 \geq \nabla(\mathcal{C}_2)$ and $\nabla(\mathcal{C}_1)$ is the largest region prevailed by $\mathcal{C}_1$, we get $\nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2)$.

Then we prove $\nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2) \Rightarrow \mathcal{C}_1 \geq \mathcal{C}_2$. From $\nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2)$, we have $\mathcal{C}_1 \geq \nabla(\mathcal{C}_1) \geq \nabla(\mathcal{C}_2) \geq \mathcal{C}_2$. So, $\mathcal{C}_1 \geq \mathcal{C}_2 \Leftrightarrow \nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2)$ and the lemma is proved. □

5.4.3 Optimising the slack-time pre-order relation

We now present another more efficient simulation relation as an extension of the slack-time pre-order relation $\succeq_{st}$ in case that tasks have constrained deadlines. Although a similar pre-order can also be defined for arbitrary-deadline sporadic tasks, for simplicity we restrict our extension only for tasks with deadlines less than or equivalent to their respective minimum interarrival times.

Let us first have a look at a simple example. Given two states s_1 and s_2 with $s_1.c_i = s_2.c_i = 0$, $s_1.p_i = T_i + 1$, $s_2.p_i = T_i + 10000$ and for any $j \neq i$, there is $s_1.c_j \geq s_2.c_j$ and $s_1.p_j \geq s_2.p_j$. Following the definition of slack-time pre-order, we would see $s_1 \not\succeq s_2$ because of $s_1.p_i < s_2.p_i$. However, for a task with $p_i \geq T_i$ that is ready for releasing a new job, the exact valuation of p_i does not really matter. In the following, we are going to extend the original slack-time pre-order given this observation.

Definition 8 (Extended slack-time pre-order)**.** *For the* SA *automaton, the extended* slack-time *pre-order* $\succeq'_{st} \subseteq \mathsf{space} \times \mathsf{space}$ *is defined as follows:* $\forall s_1, s_2$, $s_1 \succeq'_{st} s_2$ *if and only if for any* τ_i

$$s_1.p_i \geq s_2.p_i \quad \wedge \quad s_1.c_i \geq s_2.c_i$$

or

$$s_1.p_i > T_i \quad \wedge \quad s_1.c_i \geq s_2.c_i \tag{C1}$$

The extension comes from the condition in (C1): $s_1.c_i \geq s_2.c_i$ is the same as in $\succeq_{st}$; $s_1.p_i > T_i$ says that τ_i is eligible to release a new job in s_1 at any time. The extension part is meaningful when $s_2.p_i > s_1.p_i > T_i$ and in such a case the new job release of τ_i in both s_1 and s_2 can happen at any time regardless of the exact values of $s_1.p_i$ or $s_2.p_i$.

Theorem 8. *The extended pre-order relation* $\succeq'_{st}$ *is still a weak simulation relation in* $\mathsf{space}(\mathsf{SA})$.

Proof. Let us say $s_1 \succeq'_{st} s_2$. If $s_2.p_i \leq T_i$, then the original $\succeq_{st}$ pre-order relation simply holds. Otherwise, if there is $s_1.c_i = 0$, then for any job release of τ_i from s_2, τ_i can trigger its job release also in s_1; given the structure of a Task Automaton and the constraint $s_1.p_i > T_i$, there is no possibility that $s_1.c_i > 0$ (suppose that s_1 be not in a DeadlineMissed location). In the end, our proof for Theorem 6 still holds for the extended slack-time pre-order. □

Additionally, we can define the new pre-order for unconstrained-deadline tasks by modifying the condition in (C1) as: $s_1.p_i > D_i \quad \wedge \quad s_1.c_i \geq s_2.c_i$. The reasoning behind this is similar as in proof of Theorem 8 and we are not going into details of it.

As in the case of $\succeq_{st}$, we are going to adapt $\succeq'_{st}$ for symbolic state space of SA. We first extend the widening operator ∇. We define a new widening operator ∇' such that given a convex polyhedron $\mathcal{C}$, $\nabla'(\mathcal{C})$ is constructed as follows.

- For each task τ_i, if $p_i > T_i$ is satisfied but $p_i < T_i$ is not, then we unconstrain p_i; that is, after the operation there is $-\infty < p_i < +\infty$.
- Suppose $\mathcal{C}'$ be the resulting convex polyhedron after the first step; the original widening operator ∇ is then called on $\mathcal{C}'$. That is, the final widened convex region is $\nabla(\mathcal{C}')$.

Now, let us define the extended slack-time pre-order in the symbolic state space $\mathsf{Space}(\mathsf{SA})$ and prove its validity.

Definition 9. *For the* SA *automaton, the extended* slack-time *pre-order* $\succeq'_{st} \subseteq \mathsf{Space} \times \mathsf{Space}$ *is defined such that* $\forall S_1, S_2$, $S_1 \succeq'_{st} S_2$ *if and only if* $\nabla'(S_1.\mathcal{C})$ *includes* $\nabla'(S_2.\mathcal{C})$.

Theorem 9. *The extended pre-order* $\succeq'_{st} \subseteq \mathsf{Space} \times \mathsf{Space}$ *is a weak simulation relation.*

Proof. After the ∇' operation, $-\infty < p_i < +\infty$ notifies the existence of such tasks that are eligible to release new jobs at any time from then on. As in condition (5.1), the weak simulation relation between symbolic states can be derived from corresponding weak simulation relation in concrete state space. Given the definition of extended slack-time pre-order $\succeq'_{st}$ in $\mathsf{space(SA)}$, the claim in this theorem is valid. □

In the rest, we implicitly use the notation slack-time pre-order $\succeq_{st}$ to denote also the extended one.

5.4.4 Schedulability analysis in SA

In this section, we formulate the algorithm to explore the state space of SA by using a breadth-first traversal for reachability analysis and adapting the classic algorithm (Algorithm 1 Section 2.2). The pseudo-code of the Schedulability Analysis algorithm in SA (SA-SA) is shown in Algorithm 2. If some state in F is reachable, then the task set encoded in SA is deemed not-schedulable.

Algorithm 2: Schedulability Analysis in SA (SA-SA)

```
1:  R ← {S_0}
2:  while true do
3:    P ← Post(R)
4:    if P ∩ F ≠ ∅ then
5:      return NOT schedulable
6:    end if
7:    R' ← R ∪ P
8:    R' ← Max^≽(R')
9:    if R' = R then
10:     return schedulable
11:   else
12:     R ← R'
13:   end if
14: end while
```

$\mathsf{Max}^{\succeq}(R')$ is defined as

$$\forall S \in \mathsf{Max}^{\succeq}(R') : \nexists S' \in R' \ s.t. \ S' \succ_{st} S$$

$$\forall S, S' \in \mathsf{Max}^{\succeq}(R') : S' \not\succeq_{st} S$$

At line 8 of the algorithm, $\mathsf{Max}^{\succeq}$ operation eliminates from R' the states that are simulated by others. When no new state is produced (line 9), the algorithm terminates and the task set is deemed schedulable.

As long as it terminates, the correctness of Algorithm 2 can be proved following the same scheme as in [GGL13]. We skip the details here.

By replacing the $\mathsf{Max}^{\succeq}$ operator with $\mathsf{Max}^{\supseteq}$ (as defined in Section 2.2), we obtain a version of SA-SA that does not use the simulation relation. In Section 5.6.1, we will compare the efficiency of these two versions of SA-SA, with and without simulation relation. As $\succeq_{st}$ does not require the equivalence of location names between two states such that $s_1 \succeq_{st} s_2$ and convex region inclusion is a special case of convex region prevailing, there is $\mathsf{Max}^{\succeq}(R') \subseteq \mathsf{Max}^{\supseteq}(R')$. We will investigate how much efficiency improvement the schedulability analysis can obtain through simulation relation enhancement.

Unlike previous exact analysis techniques in discrete time domain, SA-SA works in continuous time domain, which makes it less sensitive to the values of task parameters. For example, given the task set $\mathcal{T}_1 = \{(C_1, D_1, T_1), \ldots, (C_n, D_n, T_n)\}$, we enlarge every task parameter by multiplying by 10 and obtain $\mathcal{T}_2 = \{(C_1 \cdot 10, D_1 \cdot 10, T_1 \cdot 10), \ldots, (C_n \cdot 10, D_n \cdot 10, T_n \cdot 10)\}$. When we apply SA-SA on $\mathcal{T}_1$ and $\mathcal{T}_2$, the number of states generated at each step will be exactly the same for the two cases, whereas this may not be true for the method in [BC07a] and [GGL13].

We have implemented SA-SA in the software FOrmal Real-Time Scheduler (FORTS)([SL14a]).

In general the reachability analysis of LHAs is not decidable. However, specific LHAs may be analysable in a finite number of steps. We now prove that the problem under investigation is indeed decidable.

5.5 The Decidability Interval

In this section, we formulate the concept of *decidability interval*, a bounded time interval starting from time 0 (that is the beginning of the schedule) such that a set of sporadic tasks is schedulable if and only if there exists no configuration of release patterns that provokes a deadline miss inside this interval. In other words, we must check the schedules generated by all possible patterns of releases, but each one only inside the decidability interval.

We then propose a formula for computing the decidability interval for G-FP scheduling of sporadic tasks with constrained deadlines, and we show that this interval is indeed quite short. Note that the concept of decidability interval is applicable to scheduling of sporadic tasks and does not rely on the system automaton SA.

In the end, based on the decidability interval obtained for G-FP scheduling, we prove the decidability of SA-SA algorithm with respect to constrained-deadline sporadic tasks. In the remainder of this chapter, by default we assume that all tasks have constrained deadlines.

5.5.1 System statuses and the dominance relation

We first introduce some preliminary concepts and properties that will be used for defining and deriving the decidability interval.

The *task status* $\delta_i(t)$ of a task τ_i at time t is defined as a pair $\delta_i(t) = (p_i(t), c_i(t))$.

- $p_i(t)$ is the *activation variable* and records the time passed since τ_i's latest activation.
- $c_i(t)$ is the *execution variable* and represents the unfinished computation of τ_i at time t.

Clearly, $p_i(t)$ and $c_i(t)$ here mimic their respective counterparts (p_i variable and c_i variable) in the task automaton TA. Please refer to Section 5.3.1 for a more detailed explanation on these variables. At any time t, there can be infinite possibilities of task statuses $\delta_i(t)$, subject to different task release patterns.

The *system status* $\Delta(t)$ at time t for the task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is defined as the composition of all its tasks' statuses, $\Delta(t) = (\delta_1(t), \ldots, \delta_n(t))$. A system status $\Delta(t)$ maps to a concrete state (l, ν) in SA, as all the values of $p_i(t)$ and $c_i(t)$ for each task: 1) on one hand represent the valuation ν in a state, and 2) on the other hand encode corresponding location information l.

Let $\mathcal{T}_k$, with $k \leq n$, be the subset containing the first k tasks from $\mathcal{T}$. We define $\Delta_k(t) = (\delta_1(t), \ldots, \delta_k(t))$ as the *partial system status* of the tasks in $\mathcal{T}_k$.

We say that a task τ_i is active under the system status $\Delta(t)$ if in the corresponding task status $\delta_i(t)$ there is $c_i(t) > 0$, and $\mathsf{A}_{\Delta(t)}$ denotes the set of active tasks under system status $\Delta(t)$. If at time t there is $p_i(t) = D_i$ and $c_i(t) > 0$, then τ_i misses its deadline.

If a task τ_i is released at time t, then task statuses for τ_i before and after the release are also different. Prior to the release, there is $p_i(t) \geq T_i$ and $c_i(t) = 0$; after the release, we have $p'_i(t) = 0$ and $c'_i(t) = C_i$. To make an explicit distinction between the two cases, we use $p'_i(t)$ and $c'_i(t)$ to denote the latter. Such a convention also applies for task status $\delta'_i(t)$ and system status $\Delta'(t)$.

(The initial configuration) We assume $\Delta(0)$ is the initial status of the system. Tasks in the system are released asynchronously and we do not know when the first instance of a task is released. Thus, we configure the initial system status as

$$\forall \tau_i \in \mathcal{T}\ p_i(0) \geq 0 \text{ and } c_i(0) = 0$$

This means that the $p_i(0)$ can be an arbitrary non-negative value.
(The dominance relation) Given two task statuses $\delta_i(t)$ and $\delta_i(t')$, at time t and t' respectively, we say $\delta_i(t)$ *dominates* $\delta_i(t')$, denoted as $\delta_i(t) \succeq \delta_i(t')$, if the following conditions hold.

- $p_i(t) \geq p_i(t')$.
- $c_i(t) \geq c_i(t')$.

The relationship can be lifted to system status: given two system statuses $\Delta(t)$ and $\Delta(t')$, we say $\Delta(t)$ *dominates* $\Delta(t')$, denoted as $\Delta(t) \succeq \Delta(t')$, if $\forall \tau_i \in \mathcal{T}$ there is $\delta_i(t) \succeq \delta_i(t')$. The dominance relation can naturally be applied between partial system statuses.

The dominance relation between system statuses mimics the slack-time pre-order relation in system automaton SA but without the underlying LHA model. By understanding this, the following property is straightforward.

Lemma 3. *Assume there are two system statuses $\Delta(t)$ and $\Delta(t')$ at t and t' respectively and $\Delta(t) \succeq \Delta(t')$. Suppose ϵ be an arbitrary non-negative value and $\Delta(t'+\epsilon)$ be a system status at $(t'+\epsilon)$ such that from $\Delta(t')$ to $\Delta(t'+\epsilon)$*

- *there is no task activation;*
- *or, there is exactly one task activation at time $(t'+\epsilon)$.*

Then, there exists a system status $\Delta(t+\epsilon)$ such that $\Delta(t+\epsilon) \succeq \Delta(t'+\epsilon)$.

Proof. We first consider the case that from $\Delta(t')$ to $\Delta(t'+\epsilon)$ there is no new task release. As tasks are activated sporadically, there exists a system status $\Delta(t+\epsilon)$ such that from $\Delta(t)$ to $\Delta(t+\epsilon)$ there is also no new task arrival. In such a case, all activation variables continuously increase. For any task τ_i there is $p_i(t+\epsilon) = p_i(t) + \epsilon$ and $p_i(t'+\epsilon) = p_i(t') + \epsilon$. As $\Delta(t) \succeq \Delta(t')$, for any task τ_i, there is $p_i(t) \geq p_i(t')$ and $c_i(t) \geq c_i(t')$; this further implies that $p_i(t+\epsilon) \geq p_i(t'+\epsilon)$.

Now, it is the turn to analyze the execution variables. If τ_i is not an active task in $\Delta(t')$, as there is no new task arrival, there will be $c_i(t') = c_i(t'+\epsilon) = 0$; thus, $c_i(t+\epsilon) \geq c_i(t'+\epsilon)$ trivially holds. On the other hand, if τ_i is indeed an active task in $\Delta(t')$, then it is also an active task in $\Delta(t)$. Suppose π and π' be the higher priority interference that τ_i suffers from t to $(t+\epsilon)$ and from t' to $(t'+\epsilon)$ respectively. Since $\Delta(t) \succeq \Delta(t')$, $\mathsf{A}_{\Delta(t)} \supseteq \mathsf{A}_{\Delta(t')}$ and for any task $\tau_j \in \Delta(t')$ its unfinished execution in $\Delta(t)$ is no smaller than its unfinished execution in $\Delta(t')$ as $c_i(t) \geq c_i(t')$. According to the G-FP scheduling policy, π will not be smaller than π', i.e., $\pi \geq \pi'$. Suppose that $c_i(t'+\epsilon) > 0$ (otherwise, $c_i(t+\epsilon) \geq c_i(t'+\epsilon)$ will trivially hold). There is actually $c_i(t'+\epsilon) = c_i(t') - (\epsilon - \pi') = c_i(t) + \pi' - \epsilon$; on the other side, $c_i(t+\epsilon) = c_i(t) + \pi - \epsilon \geq c_i(t'+\epsilon)$. As a result, we have $\Delta(t+\epsilon) \succeq \Delta(t'+\epsilon)$.

Now, let us consider the second case, that is some task τ_k is activated task at time $(t'+\epsilon)$. That is, before τ_k is released, there is $p_k(t'+\epsilon) \geq T_k$; after τ_k is activated, there is $p'_k(t'+\epsilon) = 0$ and $c'_k(t'+\epsilon) = C_k$. From t' (t) to $(t'+\epsilon)$ $((t+\epsilon))$, tasks that are different from τ_k can be analysed in the same way as in the above discussion, and we are going to concentrate on the task τ_k.

As $\Delta(t) \succeq \Delta(t')$, there is $p_k(t) \geq p_k(t')$. From t to $(t+\epsilon)$, if τ_k does not release a new task instance, there is $p_k(t+\epsilon) = p_k(t) + \epsilon \geq p_k(t') + \epsilon \geq T_k$. So, at time $(t+\epsilon)$, a new instance from τ_k is eligible to

be released, which will result in $p'_k(t+\epsilon) = 0 = p'_k(t'+\epsilon)$ and $c'_k(t+\epsilon) = C_k = c'_k(t'+\epsilon)$. As a result, $\Delta'(t+\epsilon) \succeq \Delta'(t'+\epsilon)$.

Hence, the lemma is proved. □

Then, we generalise the above Lemma 3 as follows.

Theorem 10. *Suppose $\Delta(t)$ and $\Delta(t')$ are two system statuses such that $\Delta(t) \succeq \Delta(t')$. Then, for any non-negative value Θ and for any system status $\Delta(t'+\Theta)$, there exists a system status $\Delta(t+\Theta)$ such that $\Delta(t+\Theta) \succeq \Delta(t'+\Theta)$.*

Proof. From $\Delta(t')$ to $\Delta(t'+\Theta)$, the time interval $[t', t'+\Theta]$ can be divided into a finite number (let us say N) of successive sub-intervals $[t', t'+\epsilon_1], [t'+\epsilon_1, t'+\epsilon_2], \ldots, [t'+\epsilon_{N-1}, t'+\Theta]$ such that in each such sub-interval there is no new task arrival or there is exact one new task arrival and it happens at the end of the sub-interval. Note that in case there are multiple task arrivals at a single time point, we can build multiple sub-intervals with length 0. Then, Lemma 3 can be applied successively to each sub-interval and the Theorem is proved. □

5.5.2 The decidability interval for G-FP scheduling

First, we formally define the decidability interval for a set of sporadic tasks under G-FP scheduling as follows.

Definition 10 (Decidability interval)**.** *The decidability interval is defined as a time interval $[0, L]$ such that the schedulability of task set $\mathcal{T}$ on m processors under a G-FP scheduler can be decided inside it. That is, the task set $\mathcal{T}$ is schedulable if and only if no task will miss its deadline in the interval $[0, L]$.*

In order to compute the decidability interval for G-FP scheduling, we need to introduce another concept called *dominant interval*.

Definition 11 (Level-k dominant interval)**.** *For the task subset $\mathcal{T}_k \subseteq \mathcal{T}$, its level-$k$ dominant interval is defined as a time interval $[0, L_k]$ such that for any t' and for any system status $\Delta_k(t')$, there exists $t \in [0, L_k]$ and there exists a system status $\Delta_k(t)$ such that $\Delta_k(t) \succeq \Delta_k(t')$.*

A level-k dominant interval must be a decidability interval for $\mathcal{T}_k$. Suppose that a task $\tau_i \in \mathcal{T}_k$ misses its deadline at time t' under system status $\Delta_k(t')$, i.e., $p_i(t') = D_i$ and $c_i(t') > 0$. Given the definition of a level-k dominant interval, there exists $\Delta_k(t)$ with $t \in [0, L_k]$ such that $\Delta_k(t) \succeq \Delta_k(t')$. That is, $p_i(t) \geq p_i(t')$ and $c_i(t) \geq c_i(t')$; this implies that τ_i also misses its deadline at time t. The decidability interval and level-k dominant interval can be bridged through the following two theorems.

Theorem 11. *Let $[0, L_k]$ be the level-k dominant interval for $\mathcal{T}_k$. Then, $[0, L_k + D_{k+1}]$ is the decidability interval for $\mathcal{T}_{k+1}$.*

Proof. Task τ_{k+1} cannot interfere the execution of higher priority ones from $\mathcal{T}_k$, and $\mathcal{T}_k$ is schedulable if and only if no task from it misses a deadline in the interval $[0, L_k] \subset [0, L_k + D_{k+1}]$.

As for τ_{k+1}, we will show that if there is a deadline miss for it after the time point $(L_k + D_{k+1})$, there exists also a deadline miss for τ_{k+1} inside the interval $[0, L_k + D_{k+1}]$.

Since L_k is the level-k dominant interval, for any time $t' > L_k$ and for any $\Delta_k(t')$, there exists time $t \in [0, L_k]$ and there exists $\Delta_k(t)$ such that $\Delta_k(t) \succeq \Delta_k(t')$. Suppose that task τ_{k+1} is released at an arbitrary time $t' > L_k$, after which there is $p'_{k+1}(t') = 0$ and $c'_{k+1}(t') = C_{k+1}$. If we consider the time $t \in [0, L_k]$ with $\Delta_k(t) \succeq \Delta_k(t')$, as tasks are sporadically activated and at the initial time the activation variable can have any value, then there exists a situation in which $p_{k+1}(t) \geq T_{k+1}$; this means the

task τ_{k+1} can also be activated at time t. After τ_{k+1} is released, there will be also $p'_{k+1}(t) = 0$ and $c'_{k+1}(t) = C_{k+1}$.

Thus, for any release of τ_{k+1} at any $t' > L_k$, there also exists the release for τ_{k+1} at some time $t \in [0, L_k]$ such that after the release $\Delta'_{k+1}(t) \succeq \Delta'_{k+1}(t')$. Suppose that there exists $\Delta(t' + D_{m+1})$ such that τ_{k+1} misses its deadline, i.e., $p_{k+1}(t' + D_{k+1}) = D_{k+1}$ and $c_{k+1}(t' + D_{k+1}) > 0$; according to Theorem 10, there also exists $\Delta_{k+1}(t + D_{k+1})$ with $\Delta_{k+1}(t + D_{k+1}) \succeq \Delta_{k+1}(t' + D_{k+1})$, which implies $p_{k+1}(t + D_{k+1}) = D_{k+1}$ and $c_{k+1}(t + D_{k+1}) \geq c_{k+1}(t' + D_{k+1}) > 0$, i.e., τ_{k+1} also misses its deadline at time $(t + D_{k+1})$.

For any release of τ_k at time $[0, L_k]$, its absolute deadline is bounded by $(L_k + D_{k+1})$. That is, the task τ_{k+1} is schedulable if and only if there is no deadline miss in the interval $[0, L_k + D_{k+1}]$. In conclusion, $[0, L_k + D_{k+1}]$ is the decidability interval for $\mathcal{T}_{k+1}$. □

The above Theorem 11 demonstrates how to derive a decidability interval from a dominant interval. On the contrary, the next theorem shows that a decidability interval can itself be also a dominant interval as long as the following property is verified.

Theorem 12. *The decidability interval* $[0, L_k + D_{k+1}]$ *in Theorem 11 is a level-*$(k+1)$ *dominant interval if* τ_{k+1} *is schedulable.*

Proof. We are going to discuss two cases given the task status $\delta_{k+1}(t')$ of τ_{k+1} at time $t' > L_k + D_{k+1}$: $c_{k+1}(t') = 0$ or $c_{k+1}(t') > 0$.

(Case 1: $c_{k+1}(t') = 0$) Because that L_k is the level-k dominant interval, there exists $t \in [0, L_k]$ and $\Delta_k(t)$ such that $\Delta_k(t) \succeq \Delta_k(t')$. As for τ_{k+1}, given the initial configuration and the sporadic activation behaviour, there exists task status such that $p_{k+1}(t) \geq p_{k+1}(t')$ and $c_{k+1}(t) = 0 = c_{k+1}(t')$. As a result, $\Delta(t)_{k+1} \succeq \Delta_{k+1}(t')$.

(Case 2: $c_{k+1}(t') > 0$) For such a system status $\Delta_{k+1}(t')$ with $c_{k+1}(t') > 0$, the corresponding τ_{k+1} is released at time $(t' - p_{k+1}(t'))$, denoted as t'_0. Before the release, there must be $p_{k+1}(t'_0) \geq T_{k+1}$; after the release, there is $p'_{k+1}(t'_0) = 0$ and $c'_{k+1}(t'_0) = C_{k+1}$. Then, there exists $t_0 \in [0, L_k]$ such that $\Delta_k(t_0) \succeq \Delta_k(t'_0)$ and $p_{k+1}(t_0) \geq p_{k+1}(t'_0)$ and $c_{k+1}(t_0) = 0$; this implies, by triggering the release of τ_{k+1}, $\Delta'_{k+1}(t_0) \succeq \Delta'_{k+1}(t'_0)$. According to the Theorem 10, after $p_{k+1}(t')$ time elapsing, there must exist system status $\Delta_{k+1}(t_0 + p_{k+1}(t')) \geq \Delta_{k+1}(t')$. Let us denote with $t = t_0 + p_{k+1}(t')$. As τ_{k+1} is schedulable and $c_{k+1} > 0$, there is $p_{k+1}(t') < D_{k+1}$; together with $t_0 \in [0, L_k]$, we have $t \in [0, L_k + D_{k+1})$.

In conclusion, $\forall t' > L_k + D_{k+1} \forall \Delta_{k+1}(t')$, there exists $t \in [0, L_k + D_{k+1}]$ and $\Delta_{k+1}(t)$ such that $\Delta_{k+1}(t) \succeq \Delta_{k+1}(t')$. That is, $[0, L_k + D_{k+1}]$ is a level-$(k+1)$ dominant interval. □

Theorem 11 and Theorem 12 provide a recursive way to construct the decidability interval for a given task set $\mathcal{T}$. However, the starting point of such a procedure is still missing. The following theorem fills in the gap.

Lemma 4. *The level-m dominant interval is* $[0, L_m]$ *with* $L_m = \max_{\tau_i \in \mathcal{T}_m} \{C_i\}$.

Proof. Suppose that $m = 2$ and $\mathcal{T}_2 = \{\tau_1, \tau_2\}$.

Given any time point $t' > L_2$, the system status $\Delta(t')$ is in one of the following situations.

- Both τ_1 and τ_2 are active such that $0 \leq p_1(t') < C_1$, $c_1(t') = C_1 - p_1(t')$ and $0 \leq p_2(t') < C_2$, $c_2(t') = C_2 - p_2(t')$.
- τ_1 is active and τ_2 is not active such that $0 \leq p_1(t') < C_1$, $c_1(t') = C_1 - p_1(t')$ and $p_2(t') \geq 0$, $c_2(t') = 0$.
- τ_1 is not active and τ_2 is active such that $p_1(t') \geq 0$, $c_1(t') = 0$ and $0 \leq p_2(t') < C_2$, $c_2(t') = C_2 - p_2(t')$.

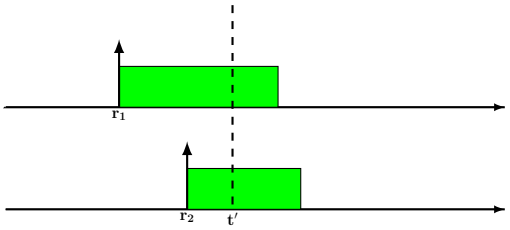


Figure 5.5: Both τ_1 and τ_2 are active

- Both τ_1 and τ_2 are not active such that $p_1(t') \geq 0$, $c_1(t') = 0$ and $p_2(t') \geq 0$, $c_2(t') = 0$.

For any of the cases listed above, it is not difficult to find a time point $t \in [0, L_m]$ with a configuration $\Delta(t)$ such that $\Delta(t) \succeq \Delta(t')$. For example, in case both τ_1 and τ_2 are active, the corresponding scenario is depicted in Figure 5.5, where τ_1 and τ_2 are released at time r_1 and r_2 respectively. At time t', there is $c_1(t') = C_1 - t' + r_1$, $p_1(t') = t' - r_1$, $c_2(t') = C_2 - t' + r_2$, $p_2(t') = t' - r_2$. Note that $p_1(t') = t' - r_1$ and $p_2(t') = t' - r_2$ are upper bounded by $L_m = \max_{\tau_i \in \mathcal{T}_m} \{C_i\}$. Then, let us consider the time $t \in [0, L_m]$ such that $t = \max\{t' - r_1, t' - r_2\}$. There exists the task release of τ_1 at time $(t - p_1(t'))$ and the task release of τ_2 at $(t - p_2(t'))$ such that at time t we will see $p_1(t) = p_1(t')$ and $p_2(t) = p_2(t')$. This implies that $c_1(t) = C_1 - p_1(t) = C_1 - p_1(t') = c_1(t')$ and $c_1(t) = C_1 - p_1(t) = C_1 - p_1(t') = c_1(t')$. As a result $\Delta(t) \succeq \Delta(t')$.

In case there is an active task (let us say τ_1) and an inactive task (let us say τ_2, i.e., $p_2(t') \geq 0$ and $c_2(t') = 0$) at time t', for the active task the above analysis still can be applied, and the time $t \in [0, L_m]$ can be found such that $c_1(t) = c_1(t')$ and $p_1(t) = p_1(t')$. For the inactive task τ_2, we remind that, at time 0, $c_2(0) = 0$ and $p_2(0)$ can be any non-negative value; when there is no new task arrival, an activation variable will monotonically increase. This means that at time t, there must exist a task status $\delta_2(t)$ for τ_2 with $p_2(t) \geq p_2(t')$ and $c_2(t) = 0$. As a result, $\Delta(t) \succeq \Delta(t')$.

In the last case, both tasks are inactive. There exists $\Delta(0)$ such that $\Delta(0) \succeq \Delta(t')$. Moreover, if we consider a general $m \geq 2$, it is simply a matter of expanding the list for combinations of active and inactive tasks. □

Starting from the level-m dominant interval and by repeatedly applying Theorems 11 and 12, we are able to compute the decidability interval for any task set $\mathcal{T}$.

Theorem 13. *For a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ with n tasks running on m processors, its decidability interval is $[0, L]$ with $L = \sum_{1 \leq i \leq m} C_i + \sum_{m < i \leq n} D_i$.*

Proof. Suppose that $\mathcal{T}$ is not schedulable. Let us say τ_k is the highest priority task among all non-schedulable tasks. According to Theorem 11, Theorem 12 and Lemma 4, the decidability interval for $\mathcal{T}_k$ is $[0, \sum_{1 \leq i \leq m} C_i + \sum_{m < i \leq k} D_i]$, which is contained in $[0, L]$. This implies that if $\mathcal{T}$ is not schedulable, then there exists necessarily a configuration of releases such that a deadline miss happens in the interval $[0, L]$. Thus, $[0, L]$ is the decidability interval for $\mathcal{T}$. □

Given the fact that a task will never have its job run for more than its WCRT, we can refine the formulation of a decidability interval.

Lemma 5. *For a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ with n tasks running on m processors, its decidability interval can be further refined as $[0, L]$ with $L = \sum_{1 \leq i \leq m} C_i + \sum_{m < i \leq n} \min\{R_i, D_i\}$.*

Proof. By replacing D_i with $\min\{R_i, D_i\}$, the proofs for Theorem 11 and Theorem 12 still hold. In the end, the new decidability interval length is valid. □

Discussion A concept called *feasibility interval* has been extensively studied in [CG06], [CG07], and [CGG11] for multiprocessor scheduling of periodic tasks. For a set of periodic tasks, its feasibility interval is a finite interval such that if all jobs released within it can meet their deadlines, then the system is schedulable. Such a result benefits from the determinism of tasks' periodic activations and its complexity is sensitive to the tasks' hyperperiod.

On the other hand, the concept of feasibility interval cannot be applied to multiprocessor scheduling of sporadic tasks, and we are not aware of any work for bounding a time interval for exact multiprocessor schedulability analysis of sporadic tasks.

The decidability interval proposed in this work is the first result that proposes a *small* bounded interval to check the schedulability of a set of sporadic tasks in global multiprocessor scheduling. Moreover, as shown in Theorem 13, the computed decidability interval for a task set can be much shorter than its counterpart for multiprocessor periodic tasks, as it does not rely on the value of hyperperiod.

5.5.3 Decidability for SA-SA algorithm

Now, we are going to answer the decidability question (in the end of Section 5.4.4) for the SA-SA algorithm. Thanks to the decidability interval computed by Theorem 13, we know that in case of a set of constrained-deadline tasks, it is enough to perform the reachability analysis of DeadlinMissed location in the bounded time interval $[0, L]$.

The reachability analysis of a generic LHA is undecidable [ACHH93], even in bounded time [BDG$^+$11]. However, [BDG$^+$11] proposed a subclass of LHA, subject to well defined language restrictions, for which the reachability problem in bounded time is decidable. More specifically, clock variables must all be monotonically increasing or stopped. The class of Stopwatch Timed Automata falls in this category.

Since our model is equivalent to a Stopwatch Timed Automaton (with an appropriate transformation of variables), it can be seen that the reachability problem of SA in bounded time is also decidable.

Theorem 14. *The termination of Algorithm 2 (SA-SA) is guaranteed in case of sporadic tasks with constrained deadlines.*

Proof. It follows from Theorem 13 in our work and from Theorem 1 in [BDG$^+$11]. □

5.6 Evaluation

In this section we evaluate the run-time performance of SA-SA by applying the algorithm to randomly generated schedulability problems. Each task set in the experiment is characterised by a tuple (m, n, U), where $m = 2$ is the number of processors, $n \in \{5, 6\}$ is the number of tasks in the task set and U is the total utilisation of the task set (i.e. $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$). Given the number of tasks n and the total task set utilisation U, the utilisation of each task is generated according to Randfixedsum algorithm ([ESD10]). Task periods are distributed in the range $[100, 1000]$. After selecting a task period T_i, the WCET is computed as $C_i = T_i \cdot U_i$; the relative deadline is then randomly sampled from C_i to T_i, After a task set is randomly generated, priorities are assigned to its tasks by the Deadline Monotonic strategy; that is, a task with shorter deadline is assigned higher priority.

Discussion It is important to underline the advantages of our methodology with respect to the methods proposed in [BC07a], [GGL13], [BMS12]. These methods typically assume task parameters such as WCETs, deadlines and periods to be rather small integers. For example, for testing their method, [GGL13] restricted tasks to have period no larger than 8. This is mainly due to the fact that they use discrete time model checking, with integer time values. Thus, these methods are sensitive to the

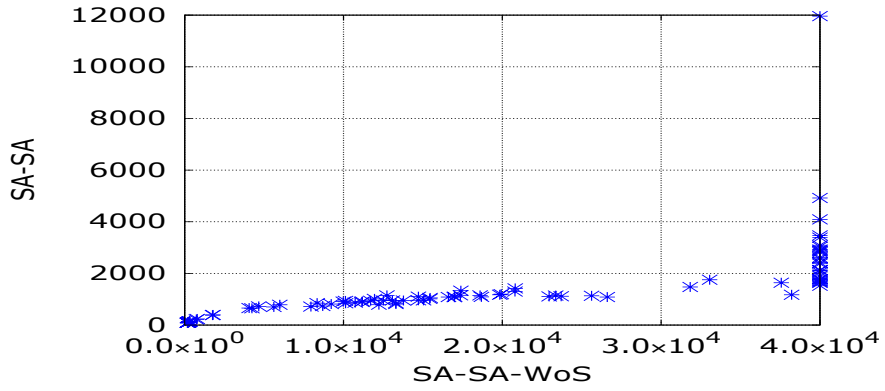


Figure 5.6: SA-SA v.s. SA-SA-WoS

absolute values of task parameters. On the other hand, relying on the formalism of LHA in continuous time domain, we are able to apply exact schedulability test on more general task configurations.

5.6.1 SA-SA algorithm with and without slack-time pre-order relation

First, we demonstrate how much the slack-time pre-order ($\succeq_{st}$) can benefit the reachability analysis in system automaton SA, by comparing the SA-SA algorithm and the SA-SA Without weak Simulation (SA-SA-WoS). We randomly generate 100 task sets with $m = 2$, $n = 5$ and utilisation randomly chosen in $[0.5, 2.0]$. Then we apply SA-SA and SA-SA-WoS to each task set, and we respectively record state space size for the two to decide the schedulability of every task set. Without the enhancement of slack-time pre-order, SA-SA-WoS faces the danger of going out of memory for some task sets. So, we put an upper threshold of 40000 for the symbolic state space size during the test of a task set by SA-SA-WoS. When the number of generated states exceeds this threshold, we stop the analysis and return with error.

Results are reported in Figure 5.6. Each point in the graphic represents a task set: the x-axis value records the number of states generated by SA-SA-WoS for checking the schedulability, and the y-axis is the state space size generated by SA-SA. As shown in the figure, by employing the slack-time weak simulation relation $\succeq_{st}$, the size of state space for schedulability check is reduced significantly. In most cases, state space sizes resulted by SA-SA and SA-SA-WoS can differ by an order of magnitude.

5.6.2 Run-time complexity of SA-SA algorithm

In Theorem 14, we have proved the termination of schedulability analysis by SA-SA. In this part, we will evaluate the factors that can affect SA-SA's run-time performance.

All simulations are conducted in a MacBook with 2.5 GHz Intel Core i5 and 8 GB memory. At first, we consider 2 processors ($n = 2$) and 5 tasks ($m = 5$). More specifically, we randomly generate a series of task sets with U uniformly distributed within $[0.5, 2]$.

By applying SA-SA on these task sets, we record the time consumed and final state space size for each test.

The results are shown in Figure 5.7a that displays the state space size of all tests by SA-SA. Each point in the figure is a task set, which is further distinguished by being schedulable and not schedulable. The x-axis denotes the utilisation of the task set and the y-axis counts the final state space size by SA-SA to decide its schedulability. Similarly, Figure 5.7b reports the time cost (in minutes) to decide a task set's schedulability and Figure 5.7c shows the relation between state space size and time cost.

As we can observe, most tests terminate in a short time with a relatively small state space size. However, with an increase of the task set utilisation, we may experience cases with a rather high run-time cost, with respect to states generated and time spent. This is due to the fact that a higher task set utilisation has a larger chance to result in a longer decidability interval (Lemma 5), within which there can be more complex task execution interleavings, thus complicating SA-SA's reachability analysis.

Furthermore, we run simulations with $m = 2$ and $n = 6$. This time, we fix several different utilisation levels for generating task sets. The time spent and state space size on each task set by SA-SA are plotted in Figure 5.8. Note that we manually stop the procedure when analysis time exceeds 7 hours (upper bound on the maximum time cost we experienced for 5 tasks running on 2 processors), and we use black colors to denote these cases in Figure 5.8a. In the same figure, we use red colors and blue colors for schedulable and unschedulable task sets respectively. Still, a task set with higher utilisation tends to complicate the analysis by SA-SA, and this is compatible with our observation from $m = 2$ and $n = 5$. On the other side, we must understand that for the unschedulable task set, SA-SA may terminate without exploring the complete state space, as long as the corresponding deadline miss condition is encountered. Thus, the schedulability check of an unschedulable task set may finish very soon. When it comes to the simulation results, as the total utilisation keeps increasing, there will more unschedulable task sets. In fact, the average run-time performance (time cost and state space size) of SA-SA improves in the high utilisation end of our simulations, as shown in Figure 5.8b and Figure 5.8c.

5.6.3 Comparison with state-of-the-art over-approximate approach

In this part, we assess the pessimism of state-of-the-art over-approximate schedulability test for sporadic tasks under G-FP policy. The analytic test we use is the Response Time Analysis with Carry-in Enumeration (RTA-CE), which will be introduced in Chapter 7. Although being the most accurate over-approximate schedulability test for G-FP scheduling problem to date, RTA-CE is still pessimistic. That is, RTA-CE may judge a schedulable task set as not-schedulable. It is interesting and meaningful to see how much gap there is between the approximate result of RTA-CE and the exact result of SA-SA.

We pick up the task set utilisation U from the set $\{0.5, 0.6 \ldots, 1.5, 1.6\}$ with $m = 2$ and $n = 5$. Then, for each U we generate 100 task sets, for which RTA-CE and SA-SA are applied. The number of schedulable tasks discovered by RTA-CE and SA-SA on each utilisation level are recorded respectively. Results are plotted in Figure 5.9. The x-axis shows the utilisation and the y-axis represents the percentage of schedulable tasks over the total number of randomly sampled tasks (at each utilisation level).

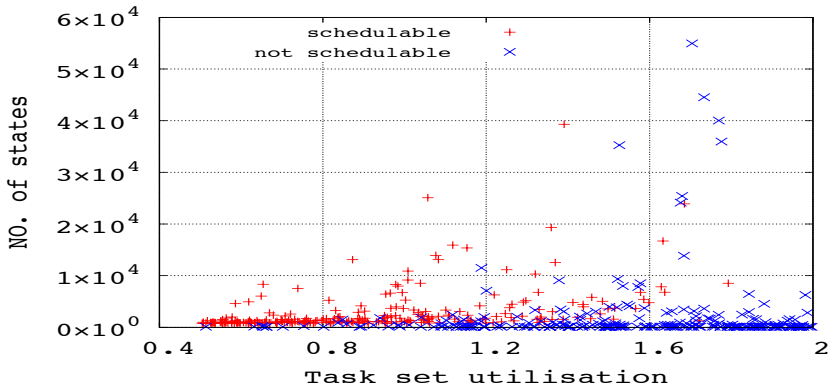
As shown in Figure 5.9, there is still a considerable number of schedulable task sets that RTA-CE failed to find. Such a gap can be seen as a motivation to further develop more precise approximate schedulability analysis.

5.6.4 Exact schedulability analysis for periodic tasks in G-FP

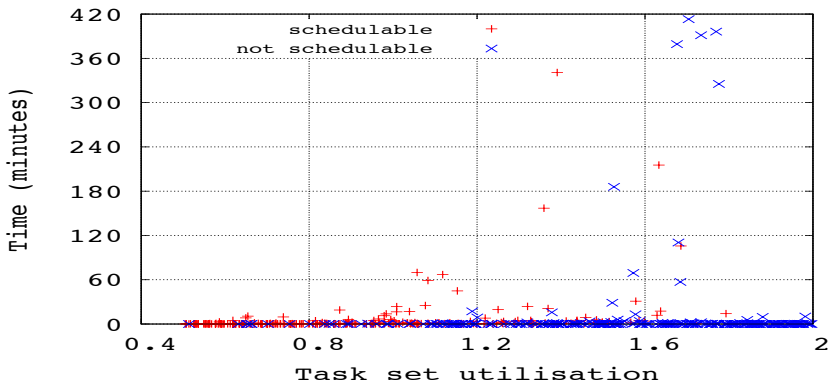
Finally, we would like to discuss the difference in complexity between exact analysis of *periodic tasks* and *sporadic tasks*.

In the case of single processor, for the fixed-priority scheduling, the schedulability analyses of sporadic tasks and synchronous periodic tasks are not really different, as they share the same worst-case scenario. When it comes to the multiprocessor, no worst-case scenario has even been found for sporadic or periodic tasks under G-FP. Therefore, to perform an exact analysis we need to analyse all possible task activations and interleavings, for both sporadic and periodic tasks.

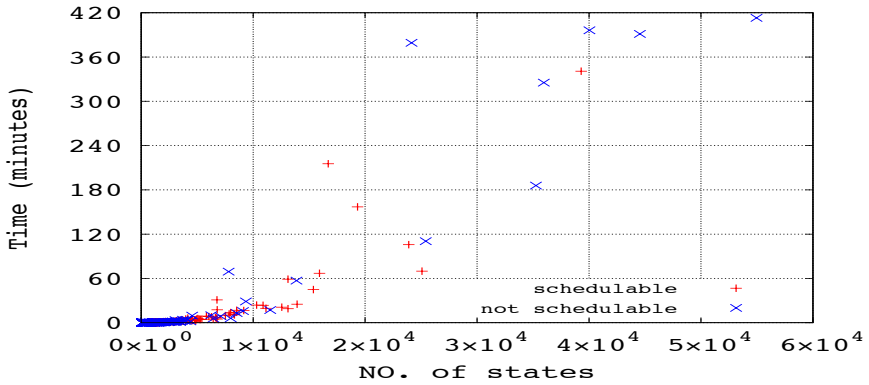
As we proved in Theorem 13, the exact analysis of sporadic tasks can be done in a decidability interval, which is relatively small. Therefore, the complexity of analysing sporadic tasks comes from the non-deterministic activations, which produce many different arrival patterns that need to be analysed.



(a) State space size

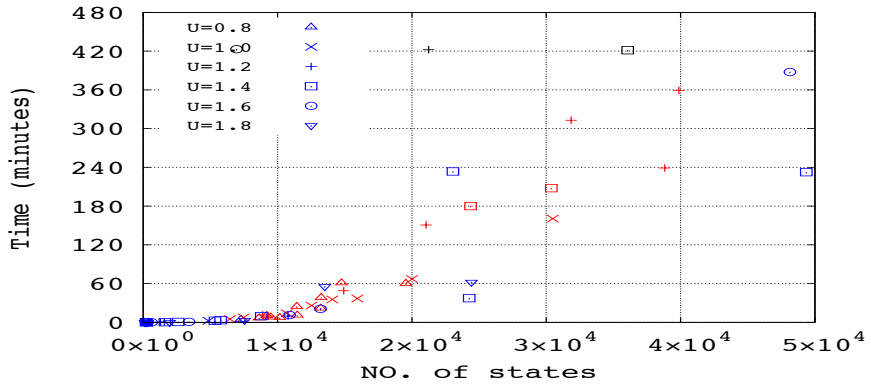


(b) Time cost

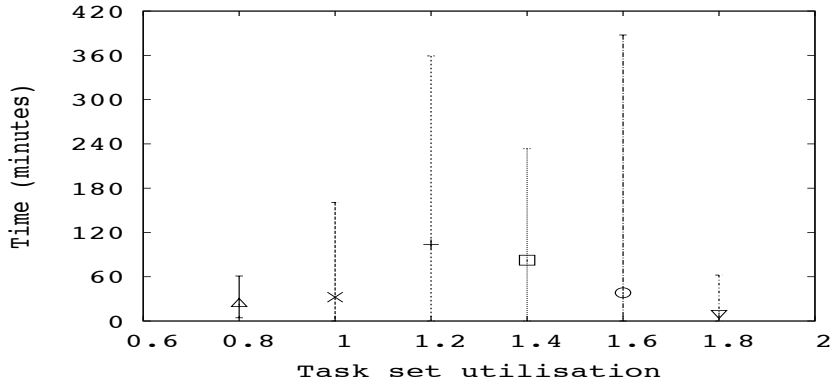


(c) State space size v.s. Time cost

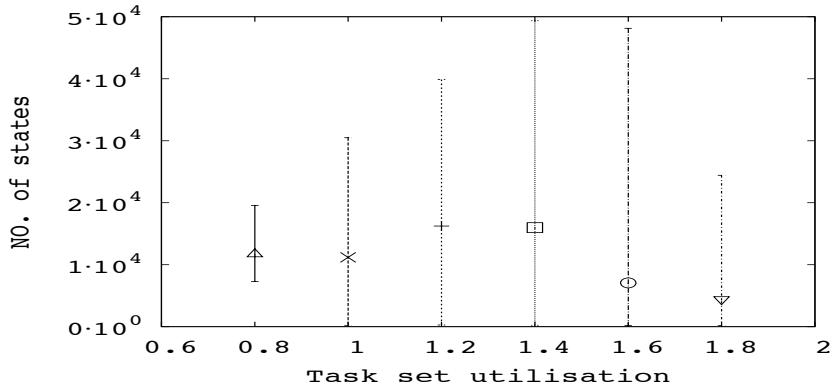
Figure 5.7: Run-time complexity of SA-SA for $m = 2$ and $n = 5$



(a) State space size v.s. Time cost



(b) Average time cost



(c) Average state space size

Figure 5.8: Run-time complexity of SA-SA for $m = 2$ and $n = 6$

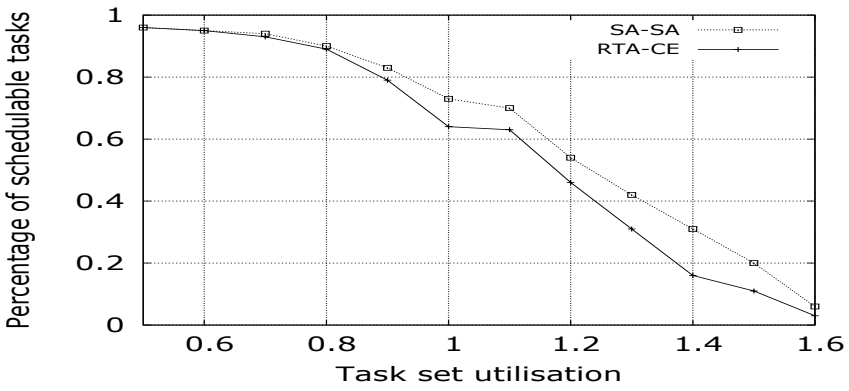


Figure 5.9: Comparison between RTA-CE and SA-SA

As for periodic tasks, they have deterministic activation patterns, and the complexity of an exact analysis is affected by the length of the feasibility interval, which is directly related to the hyperperiod of the tasks.

In this section, we refer to [GGD+07] and [GGL+08] for the exact test for periodic tasks. Both works adopt the discrete approach to model the schedule of a set of periodic tasks running on multiprocessors. Even though [GGD+07] builds the model by Timed Automata, the continuous time is discretised by using a global clock. Thus, they restrict the absolute values of task parameters to small integers. Actually, in both cases the authors choose task periods in a range $[8, 20]$ for their experiments.

In principle, the analysis scheme for periodic tasks considers all possible combinations of tasks' initial offsets; for an arbitrary combination of initial offsets, tasks' periodic activations are fixed, the system is simulated and any deadline miss is checked.

In the following, we replicate the experiment in [GGL+08], where there are 8 tasks running on 4 processors. Our goal here is not to solve the exact schedulability analysis of periodic tasks, but rather to demonstrate the different run-time complexity. Therefore, we consider all offsets to be equal to 0, so all tasks arrive at the same time.

We randomly generate 1000 periodic task sets. The system automaton for schedulability check for periodic tasks can be generated similarly as the SA for sporadic tasks: in Figure 5.2, the invariant in location Idle is set to $p \leq T$. We also restrict ourselves to tasks with constrained deadline (i.e. $D \leq T$), so we remove all guards and actions from the transitions from Waiting to itself, and from Running to itself. Finally, we do not use our simulation relation, because it is only valid for sporadic tasks and cannot be easily extended to periodic tasks.

In the end, we use FORTS to check the schedulability of each task set and records the time cost for deciding the schedulability. Results are reported in Figure 5.10. The x-axis denotes the length of the hyperperiod of a task set and the y-axis denotes the size of the state space when schedulability check is completed for the task set. As we anticipated, the number of states to explore increases with the length of hyperperiod. The maximum time we experienced for deciding a task set's schedulability is less than 3 minutes. It follows that, deciding the schedulability of a set of periodic tasks is in average *easier* than checking the schedulability of a set of sporadic tasks.

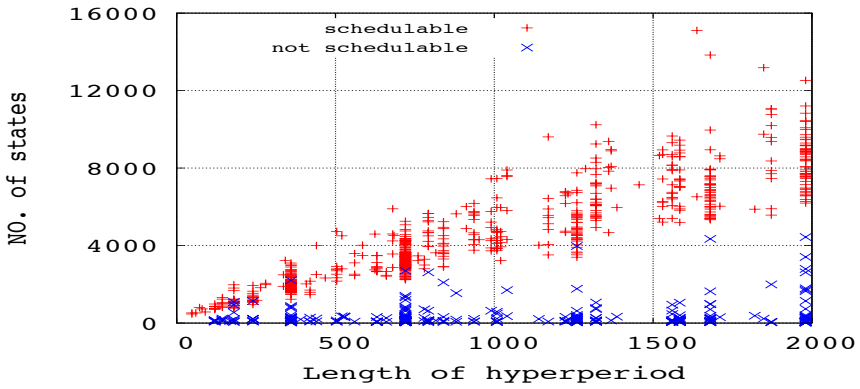


Figure 5.10: The exact analysis for periodic tasks

5.7 Conclusion

In this chapter, we formally model the exact G-FP scheduling on a multiprocessor platform; we propose a weak simulation relation for reducing the complexity of state space exploration. Then, we formulate SA-SA algorithm for the exact schedulability test. Even though the reachability problem in LHA is undecidable, we prove the correctness and termination condition for SA-SA. Compared to previous works on exact analysis, our methodology allows more complex task sets: we are able to analyse tasks with arbitrary values of parameters. On the other hand, our simulation relation work can be regarded as a general approach to mitigate the complexity brought by sporadic events when modeling real-time systems with a formalism like Stopwatch Timed Automata.

However, as task set size increases, the complexity of exact analysis is still too high even with our approach. We are working on other ways of reducing the complexity: first, we would like to use a different representation for the symbolic states (e.g. Octagons [Min06] or Difference Bound Matrices [Min01]), which requires an approximate analysis similar to the ones used for Stopwatch Timed Automata and Time Petri Nets [BFSV04]. Second, we are investigating the possibility to enhance and extend our simulation relation, so to further reduce the state space.

Furthermore, although exact critical instants for G-FP scheduling are not known, in Chapter 7 we prove a class of release patterns that can lead to worst-case response time. By exploring the advanced study in scheduling theory, we may achieve simpler models or even faster state space analysis methods.

Finally, we proved that the schedulability analysis of a sporadic task set can be done in bounded time. To our knowledge, this is the first work to demonstrate the decidability of the problem and to formulate such an upper bound. It will be interesting to see if we can apply this result to further enhance the approximate schedulability analysis methods.

Chapter 6

Multiprocessor Global Scheduling

In this chapter, we will briefly revise the analytical schedulability analysis of a set $\mathcal{T}$ of n sporadic tasks running on m ($m < n$) processors by the multiprocessor global scheduler. While many previous works exist, we will present the results that are most relevant to ours.

We focus on two kinds of global scheduling policies: the Global Earliest Deadline First (G-EDF) and the Global Fixed Priority (G-FP). Moreover, the *discrete time domain* is assumed. That is, $r_{i,j}$, $d_{i,j}$ and $f_{i,j}$ of a task will be non-negative integers.

To simplify the expressions, the following notations are used: $[\![x]\!]_y = \max\{x, y\}$, $[\![x]\!]^y = \min\{x, y\}$ and $[\![x]\!]_y^z = [\![[\![x]\!]_y]\!]^z$.

6.1 Introduction

One important class of real-time schedulers for multiprocessor systems is the class of *global scheduling policies*. According to global scheduling, all ready tasks are enqueued in a single ready queue, and the m highest priority jobs are executing on the m processors. In global preemptive scheduling, a task executing on one processor may be preempted by a higher priority task and later resume execution on a different processor: therefore, we say that this class of scheduling policies allow *migrations* of tasks among processors. Experimental comparison in the Linux operating system shows that global/clustered configurations are a viable solution for multiprocessor platforms [LFCL12].

The two most popular global scheduling algorithms are Global Fixed Priority (G-FP) and Global Earliest Deadline First (G-EDF). In G-FP scheduling, a fixed priority is assigned to each task and all jobs of the task share the assigned priority; in G-EDF scheduling, a job's priority is decided by its associated absolute deadline, a smaller absolute deadline corresponding to a higher priority.

As we discussed in Chapter 5, any exact schedulability condition for a set of sporadic tasks scheduled by global scheduling requires to test a very high number of release patterns of the tasks. For this reason, most researchers focused on deriving approximated analyses, i.e. *sufficient* conditions for schedulability. We say that schedulability test A *dominates* another schedulability test B, denoted as A $\succeq$ B, if every task set that is deemed schedulable by B is also deemed schedulable by A, and there may exist sets of tasks which are deemed schedulable by A but not by B.

Baker [Bak03] developed a sophisticated analysis technique based on the concept of *problem window*. The technique consists in checking the schedulability of one task at a time: first, the problem window is selected equal to the interval between the arrival time and the deadline of one instance of the task under analysis; then, the *interference* of higher priority jobs is computed and taken into account in the schedulability analysis. Interfering jobs are divided into *carry-in* jobs (i.e. jobs which may start executing before the problem window and whose computation time only partially contributes to the

interference) and non-carry-in jobs (i.e. jobs whose arrival time and execution time are contained in the problem window).

Baker's technique has since been extended by many researchers who tried to improve the estimation of the interfering workload. Bertogna et al. [BCL05] discovered that for each competing task with very high workload in the problem window, the part of its workload that has to be executed in parallel with the analysed task should not be taken into account in the actual interference. Later, Bertogna and Cirinei [BC07b] applied this technique to iterative Response Time Analysis (RTA) of global scheduling, including both G-EDF and G-FP.

Another breakthrough was proposed by Baruah [Bar07]. His technique tries to limit the number of carry-in tasks to be $m-1$. Although such a technique was originally conceived for G-EDF, Guan et al. [GSYY09] combined it with the RTA technique, obtaining the more precise schedulability test for G-FP scheduling. In case of G-FP schedulability analysis, the following dominant relation holds: [GSYY09]$\succeq$[BC07b]. Instead of using the iterative procedure as in [GSYY09], [DB11] directly measures the interference within the problem window by considering limited number of carry-in tasks. Though the test in [DB11] is more pessimistic than the one in [GSYY09], it is compatible with the Optimal Priority Assignment scheme, thus has the advantage when considering priority assignment of tasks under G-FP. In [LA13], there is another G-FP test based on limiting carry-in workload, but it is incomparable with the tests just mentioned. Lee and Shin [LS13] generalised the limited carry-in idea to any work-conserving algorithms.

When it goes to G-EDF, besides the tests in [Bak03, BCL05, BC07b, Bar07], many other tests have been proposed, like [BB09a] and [BBMSS09], and none of these tests can dominate the others. However, according to empirical evaluations [BB11], tests in [Bar07] and [BC07b] are shown to have better average performance.

In [LSSE15] a compositional theory is proposed to improve the overall schedulability results, which explores the sufficient condition such that to apply existing over-approximate tests for a subset of total tasks on a subset of processors. While it is likely that the dominance relation between underlying schedulability tests can be kept also for the composition result, a further study is out of the scope of this thesis. A "divide-and-conquer" approach is proposed in [Lee14] and applied to the RTA for G-EDF in [BC07b], which additionally finds certain schedulable task sets that the original test fails to detect.

6.2 Basic Notations

In this section, we will introduce these notations that are used in the context of multiprocessor global scheduling.

Tasks are tested one by one for schedulability and we assume the *target* one is τ_k; in particular, we focus on an arbitrary job of τ_k, which is also called the target job. When analysing the schedulability of τ_k, a problem window is assumed. A *problem window* is a time interval $[a, b)$ with certain length. More detailed formulation of a problem window may depend on different tests.

The *workload* of a task τ_i in a time interval of length t is the amount of computation that τ_i requires within this time interval. The *interference* of a task τ_i on the target task τ_k in a time interval is the cumulative length of the intervals during which jobs of τ_i execute and jobs of τ_k have been released but cannot execute. Since existing schedulability tests can only provide an upper bound on the interference, we abuse the use of notations interference and interference upper bound. This also happens for workload and workload upper bound.

A task τ_i is called a *carry-in* (CI) task if at least one job of τ_i has been released before the beginning of the window and executes within the window. If no such job exists, the task is called *non-carry-in* (NC)

task. That is, tasks are differentiated into NC tasks $\mathcal{T}^{NC}$ and CI tasks $\mathcal{T}^{CI}$ such that $\mathcal{T} = \mathcal{T}^{NC} \cup \mathcal{T}^{CI}$ and $\mathcal{T}^{NC} \cap \mathcal{T}^{CI} = \emptyset$. Without or without carry-in, a task may result in different workload (interference). Thus, the workload of a task can be explicitly distinguished into NC workload and CI workload; the same rule applies for interference also.

6.3 Tests for G-EDF

As for the schedulability analysis for G-EDF, here we briefly introduce Bertogna and Cirinei's Response Time Analysis (RTA) [BC07b], denoted as BC, and Baruah's test [Bar07], denoted as Bar. Note that both tests assume sporadic tasks with constrained deadlines.

(Demand bound function) In EDF scheduling, the concept of *demand bound function* plays an important role for schedulability analysis. For any t, the demand bound function $\mathsf{DBF}_i(t)$ of a task τ_i bounds the maximum cumulative execution requirement by jobs of τ_i that have *both arrival time and deadline within any interval of length* t. It is formally defined as follows.

$$\mathsf{DBF}_i(t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i \tag{6.1}$$

For G-EDF schedulability analysis, a commonly used problem window is a time interval $[a, b)$ such that b is coincident with the target job's absolute deadline.

For a NC task τ_i, its interference in a problem window (with length t) can be bounded by using the DBF function directly:

$$I_i^{NC}(t) = \mathsf{DBF}_i(t)$$

If τ_i is a CI task, its possible interference is estimated as:

$$I_i^{CI}(t) = \left\lfloor \frac{t}{T_i} \right\rfloor \times C_i + [\![(t \bmod T_i) - D_i + R_i]\!]_0^{C_i} . \tag{6.2}$$

The worst-case CI interference corresponds to the scenario depicted in Figure 6.1. Among the sequence of jobs from τ_i that may interfere with the target job, the first job is called *carry-in job*, and the last one is called *carry-out job*. The CI task's interference in the problem window is maximised when: 1) deadline of carry-out job is coincident with b, 2) every job of τ_i is released as soon as possible, and 3) the carry-in job exactly finishes the execution with its worst-case response time. The inequality $I_i^{NC}(t) \leq I_i^{CI}(t)$ always holds.

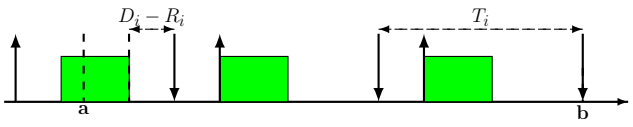


Figure 6.1: Maximum CI interference under G-EDF

6.3.1 BC

Bertogna and Cirinei [BC07b] formulated a generic upper bound to the workload of a task τ_i over an arbitrary time interval with some length L that does not depend on the specific scheduling algorithm:

$$\mathfrak{W}_i(L) = N_i(L)C_i + [\![L + R_i - C_i - N_i(L)T_i]\!]_0^{C_i}$$

where $N_i(L) = \left\lfloor \frac{L+R_i-C_i}{T_i} \right\rfloor$.

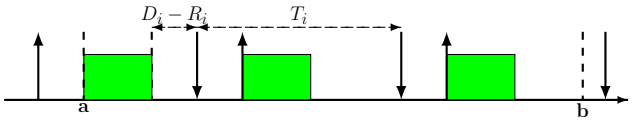


Figure 6.2: Maximum workload in a problem window

The situation that brings the worst-case workload is depicted in Figure 6.2: the carry-in job starts execution at the beginning of the window and finishes exactly with its worst-case response time; and every successive instance of τ_i arrives as soon as possible.

Finally, BC relies on a classic iterative response time analysis.

Theorem 15 (Theorem 6 in [BC07b])**.** *An upper bound on the response time of a task τ_k in a G-EDF scheduled multiprocessor system can be derived by the fixed point iteration over X of the following expression, starting with $X = C_k$.*

$$X \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} I_{i,k}(X) \right\rfloor$$

with $I_{i,k}(X) = \min(\mathfrak{W}_i(X), I_i^{CI}(D_k), X - C_k + 1)$.

The term $X - C_k + 1$ is due to the fact that if the target task is schedulable, then for any task the part of its execution that is in parallel with τ_k's execution does not result in interference [BCL05] and such a constraint is also applied to other tests for global scheduling.

For every task, its response time R_i is needed to compute $\mathfrak{W}_i(X)$ and $I_i^{CI}(D_k)$. However, R_i is unknown. This can be solved by the following iterative procedure.

1. For every task, its R_i is initialised to D_i.
2. Apply Theorem 15 to every task in the system. If the resulting response time is $> D_i$ for some task, then that task is marked as "potentially unschedulable". If there are no potentially unschedulable tasks, the task set is declared to be schedulable.
3. Repeat step (2) until there is no R_i update.

Please note that in BC all tasks are considered as CI tasks.

6.3.2 Bar

Baruah [Bar07] derived another sufficient schedulability test for G-EDF. Given the problem window $[a, b)$ with length D_k, the idea in Bar is to extend the interval back to some point s_0 at which at least one of the m processors is idle, and from s_0 to a all processors are busily executing jobs with absolute deadlines no larger than the target one's. Let define $A_k = a - s_0$. Such a time interval $[s_0, s_0 + A_k)$ is called the **busy period**. Thanks to this extension, there are at most $(m-1)$ CI tasks at time point s_0. In order for τ_k to meet its deadline, it is sufficient that all m processors are executing jobs other than τ_k for no more than $A_k + (D_k - C_k)$ time units over $[s_0, b)$.

Different from BC, Bar explicitly differentiates the interference caused by the NC task ($I_{i,k}^{NC}$) and interference caused by the CI task ($I_{i,k}^{CI}$) in the extended problem window.

$$I_{i,k}^{NC} = \begin{cases} \left[\!\left[I_i^{NC}(A_k + D_k) \right]\!\right]^{A_k + D_k - C_k + 1} & \text{if } i \neq k \\ \left[\!\left[I_i^{NC}(A_k + D_k) - C_k \right]\!\right]^{A_k} & \text{if } i = k \end{cases}$$

$$I_{i,k}^{CI} = \begin{cases} \left[\!\left[I_i^{CI}(A_k + D_k) \right]\!\right]^{A_k + D_k - C_k + 1} & \text{if } i \neq k \\ \left[\!\left[I_i^{CI}(A_k + D_k) - C_k \right]\!\right]^{A_k} & \text{if } i = k \end{cases}$$

Let us define $I_{i,k}^{DIFF} = I_{i,k}^{CI} - I_{i,k}^{NC}$, then there is the following theorem for schedulability analysis for G-EDF.

Theorem 16 (Theorem 3 in [Bar07])**.** *A task set $\mathcal{T}$ is schedulable with G-EDF if, for any $\tau_k \in \mathcal{T}$ and all $0 \leq A_k \leq \overline{A}_k$, the following holds:*

$$\Omega \leq m(A_k + D_k - C_k) \tag{6.3}$$

where Ω is the total interference:

$$\Omega = \left(\sum_{\tau_i \in \mathcal{T}} I_{i,k}^{NC} + \sum_{\textit{the } (m-1) \textit{ largest}} I_{i,k}^{DIFF} \right) \tag{6.4}$$

C_Σ denotes sum of the $(m-1)$ largest WCETs among tasks and $\overline{A}_k$ is defined as follows:

$$\overline{A}_k = \frac{C_\Sigma + D_k U_{tot} - m D_k + \sum_{\tau_i \in \mathcal{T}} (T_i - D_i) U_i + m C_k}{m - U_{tot}} \tag{6.5}$$

Additionally, the condition in Equation (6.4) needs only be tested with A_k values at which $\mathsf{DBF}_i(A_k + D_k)$ changes for some task τ_i. When $m = 1$, Bar is equivalent to the exact schedulability test for EDF in [BMR90].

6.4 Tests for G-FP

In this section, we present two state-of-the-art tests for G-FP scheduling. Also in [BC07b], similarly as the RTA for G-EDF, Bertogna and Cirinei propose the RTA for G-FP scheduling, denoted as BC-FP. Furthermore, by integrating the limited carry-in technique in Bar and the RTA in BC-FP, Guan et al. develop the RTA with Limited Carry-in (RTA-LC) for G-FP scheduling in [GSYY09]. The Deadline Analysis with Limited Carry-in (DA-LC) in [DB11] is the state-of-the-art test when regarding the priority assignment problem subject to G-FP scheduling. While BC-FP and DA-LC deal with constrained-deadline tasks, RTA-LC applies for arbitrary-deadline tasks.

To analyse the schedulability of a task τ_k, we need to take only all higher priority tasks $\tau_1, \ldots, \tau_{k-1}$ into account and compute the interference that they may cause on τ_k. As for the problem window $[a, b)$, for G-FP schedulability analysis, it is built such that at the time point a task τ_k releases its first job within the window.

6.4.1 BC-FP

The test BC for G-EDF as in Theorem 15 can be easily adapted for G-FP scheduling by ignoring the CI interference estimated in Equation (6.2), which is computed through aligning tasks' absolute deadlines.

Theorem 17 (Theorem 7 in [BC07b])**.** *An upper bound on the response time of a task τ_k in a G-FP scheduled multiprocessor system can be derived by the fixed point iteration over X of the following*

expression, starting with $X = C_k$.

$$X \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i<k} I_{i,k}(X) \right\rfloor$$

with $I_{i,k}(X) = \min(\mathfrak{W}_i(X), X - C_k + 1)$.

6.4.2 RTA-LC

The RTA-LC (Response Time Analysis with Limited Carry-in)[1]. in [GSYY09] is the start-of-the-art algorithm for computing the worst-case response time for global FP scheduling on multiprocessors. It integrates the limited carry-in idea in Bar and the response time analysis in BC-FP, and it also generalises the schedulability analysis to arbitrary-deadline tasks.

To get a safe upper bound on the WCRT of the target task τ_k, [GSYY09] proves that it is sufficient to consider the problem window $[a, b)$ such that such that there are at most $m - 1$ tasks with carry-in. Thus, we denote with $\mathcal{Z}$ the set of all possible carry-in task sets with no more than $m-1$ higher priority tasks.

Given a higher priority task τ_i and a problem window with lenght x, its workload in the window and interference upon the target task are formulated as follows. Since tasks may have unconstrained deadlines, there can be more than one active jobs from τ_k in the window and the term h is used to explicitly emphasises this number.

Lemma 6 (Lemma 2 in [GSYY09])**.** *The workload bounds can be computed with*

$$W_i^{NC}(x) = \left\lfloor \frac{x}{T_i} \right\rfloor \cdot C_i + [x \bmod T_i]^{C_i} \tag{6.6}$$

$$W_i^{CI}(x) = \left\lfloor \frac{[x - C_i]_0}{T_i} \right\rfloor \cdot C_i + C_i + \alpha \tag{6.7}$$

where $\alpha = [[x - C_i]_0 \bmod T_i - (T_i - R_i)]_0^{C_i - 1}$.

Lemma 7. *Let* τ_k *be a task under analysis, and let* τ_i *be a higher priority task. If task* τ_k *is schedulable, then an upper bound to the interference that* τ_i *causes to the first* h *instances of* τ_k *in a problem window of length* x $(x \geq h \cdot C_k)$ *is given by:*

$$I_k^{NC}(\tau_i, x, h) = \min(W_i^{NC}(x), x - h \cdot C_k + 1) \tag{6.8}$$

$$I_k^{CI}(\tau_i, x, h) = \min(W_i^{CI}(x), x - h \cdot C_k + 1) \tag{6.9}$$

Given a time interval x, the total interference $\Omega_k(x, h)$ on the first h jobs of task τ_k is defined as :

$$\max_{\mathcal{T}^{CI} \in \mathcal{Z}} \left(\sum_{\tau_i \notin \mathcal{T}^{CI}} I_k^{NC}(\tau_i, x, h) + \sum_{\tau_i \in \mathcal{T}^{CI}} I_k^{CI}(\tau_i, x, h) \right) \tag{6.10}$$

$\Omega_k(x, h)$ upper bounds the interference from higher priority tasks to the first h jobs of τ_k in the problem window. $\Omega_k(x)$ sometimes used as an abbreviation for $\Omega_k(x, 1)$. $\Omega_k(x, h)$ can be computed in linear time, since it is sufficient to find the $m - 1$ maximal values of the difference $I_k^{CI}(\tau_i, x, h) - I_k^{NC}(\tau_i, x, h)$. Given the workload formulation in Lemma 6, I_k^{CI} is never smaller than I_k^{NC}.

[1] This naming convention can be traced back to [DB11].

Theorem 18 (Theorem 2 in [GSYY09]). *For each $h \geq 1$, let $\mathcal{X}^h$ be the minimal solution of the following Equation by doing an iterative fixed point search of the right hand side starting with $x = h \cdot C_k$.*

$$x = \left\lfloor \frac{\Omega_k(x,h)}{m} \right\rfloor + h \cdot C_k \tag{6.11}$$

then,

$$R_k = \max_{h \in [1,H]} \{\mathcal{X}^h - (h-1) \cdot T_k\} \tag{6.12}$$

is an upper bound of τ_k's WCRT.

The H in (6.12) is an upper bound to the values of h that need to be checked to get the maximal response time bound R_k. The response time analysis procedure terminates as long as the obtained $\mathcal{X}^h - (h-1) \cdot T_k$ is no larger than T_k. It has been proved in [GSYY09] that a bounded H exists to guarantee the termination of the response time analysis procedure if task τ_k satisfies

$$\sum_{i<k} V_i^k + M \times U_k \neq M, \text{ where } V_i^k = \min(U_i, 1 - U_k) \tag{6.13}$$

6.4.3 DA-LC

Instead of an iterative procedure as in RTA-LC, DA-LC in [DB11] measures the total interference within a problem window having the length D_k directly.

Theorem 19 (Equation (14) in [DB11]). *For the target task τ_k and the task set $\mathcal{T}$, τ_k is schedulable if the following condition holds.*

$$D_k \geq \left\lfloor \frac{\Omega_k(D_k)}{m} \right\rfloor + C_k \tag{6.14}$$

Chapter 7

Improving the RTA for G-FP Scheduling

In this chapter we address the problem of schedulability analysis for a set of sporadic tasks with arbitrary deadlines running on a multiprocessor system with Global Fixed-Priority (G-FP) preemptive scheduling. The discrete time domain is assumed.

We prove the existence of a class of *critical instants* for releasing a task, one of which results in the worst-case response time of that task.

Then, we propose a new analysis that improves over Response Time Analysis with Limited Carry-in (RTA-LC), the state-of-the-art schedulability analysis with respect to G-FP, by reducing its pessimism. We also observe that, in the case of unconstrained deadlines, the RTA-LC may underestimate the carry-in workload, and we propose a new formulation that corrects the problem. Finally, we evaluate the performance improvement of our new response time analysis method by empirical evaluation with randomly generated task sets. Simulation results show that our new analysis method can successfully accept a considerable amount of task sets that have to be treated as unschedulable by existing methods.

The content in this chapter is based on [SLGY14].

7.1 Critical Instants for G-FP Scheduling

We first define the concept of "critical instants" for a task τ_k and we prove that the Worst-Case Response Time (WCRT) of τ_k happens when it is released precisely at one of these critical instants. This notion has already been proposed by Guan et al. [GSYY09]; later Davis et al. [DB11] independently proved that, for constrained-deadline tasks, releasing a task at one of its "critical instants" will result in the WCRT. Here we prove it directly for the general case of arbitrary deadlines.

Clearly, it makes no sense to discuss the critical instants for the m highest priority tasks, since they can execute whenever eligible and their WCRTs simply equal to their WCETs. In the following context, when using the notion of critical instants and response time analysis, we only refer to tasks other than the m highest priority ones.

Definition 12. *A critical instant for task τ_k is an instant t such that:*

- *At least m tasks with priority higher than τ_k are active at t;*
- *At most $m-1$ tasks with priority higher than τ_k are active just before t.*

The critical instant defined above does *not* precisely correspond to a concrete system state at run-time. Instead, it covers a set of possible system states, one of which will lead to the worst-case response

time as we will prove in the following. In particular, there can be up to $m-1$ higher priority tasks executing before t and we do not know which are these tasks, when these tasks have been activated and how much of their computation time is left at t.

Apart from the critical instants defined above, we also claim that the worst-case response time of a task occurs when this task releases jobs "as fast as possible", which is formally captured by the following definition:

Definition 13. *Given a sporadic task τ_k, a* chain of consecutive jobs *(or* chain *for short) $J_{k,s}, \ldots, J_{k,h}$, with arrival times $r_{k,s}, \ldots, r_{k,h}$, respectively, is defined as follows:*

- *The job preceding $J_{k,s}$ (if any) has completed before $r_{k,s}$;*
- *every job $J_{k,j}$, $j = s, \ldots, h-1$ completes after $r_{k,j+1}$.*

The chain is said to be tight *if $\forall j = s+1, \ldots h$, $r_{k,j} = r_{k,j-1} + T_k$.*

Note that every job in the system is included in some chain. For instance, in the special case of a constrained-deadline task with ($D_k \leq T_k$), every tight chain only contains one job.

Lemma 8. *The worst-case response time of τ_k is found with a job $J_{k,h}$ in a* tight *chain.*

Proof. We prove the lemma by showing that for an arbitrary chain $J_{k,s}, \ldots, J_{k,h}$ of task τ_k, if we shift the release time of every job $J_{k,j}$ in the chain to $r'_{k,j} = r_{k,s} + (j-s)T_i$, thus obtaining a tight chain, the finishing time of every job $f_{k,j}$, $j = s, \ldots, h$ does not change.

A job cannot start executing until the prior job of the same task has terminated. However, in the original chain, the finishing time of every job (except the last one) is after the arrival time of the successive job. Therefore, when job $J_{k,j}$ arrives at $r_{i,j}$, it must still wait until $f_{k,j-1}$ before it can be activated. By shifting its arrival time to $r'_{k,j} = r_{k,s} + (j-s)T_k$ this precedence relationship is maintained. Therefore, nothing in the schedule changes, and the finishing time of each job in the chain remains the same. □

We now prove the critical instant defined in Definition 13 indeed leads to the worst-case response time of a task.

Theorem 20. *The worst-case response time of τ_k is found with a job $J_{k,h}$ in a* tight chain *$J_{k,s}, \ldots, J_{k,h}$, such that $r_{k,s}$ coincides with a critical instant for task τ_k.*

Proof. First of all, by Lemma 8, if the job with the worst-case response time is not in a tight chain, we can find another job, whose response time is not smaller, in a tight chain. So in the following we only focus on jobs in tight chains.

We will show that, if worst-case response time of task τ_k occurs in a tight chain $J_{k,s}, \ldots, J_{k,h}$ that does not start at a critical instant, by modifying the arrival times of the chain we can construct a tight chain $J_{k,s}, \ldots, J_{k,h}$ starting at a critical instant, such that the new response time $R'_{k,h}$ of job $J_{k,h}$ in the resulting chain is no smaller than its counterpart $R_{k,h}$ in the original chain.

The release time of the first job in the original chain does not coincide with a critical instant, so we have two possibilities:

- Suppose that at $r_{k,s}$ there are no more than $m-1$ higher priority tasks active. Let $[A, B]$ be the first interval after $r_{k,s}$ such that there are at least m higher priority active tasks in every instant of $[A, B]$. It is easy to see that $A < f_{k,h}$, because otherwise we can set $r'_{k,s}$ and obtain a chain with a higher response time, as τ_k cannot execute in $[A, B]$.

  Therefore, τ_k executes in $[r_{k,s}, A]$. If we set $r'_{k,s} = A$, the new finishing time of $J_{k,h}$ is

$$f'_{k,h} \geq f_{k,h} + (A - r_{k,s})$$

So the new response time of $J_{k,h}$ is

$$\begin{aligned} R'_{k,h} &= f'_{k,h} - r'_{k,h} \\ &\geq f_{k,h} - (r_{k,s} - A) - r'_{k,h} \\ &= f_{k,h} - (r_{k,s} - A) - (A + (h-s)T_k) \\ &= f_{k,h} - (r_{k,s} + (h-s)T_k) = f_{k,h} - r_{k,h} = R_{k,h} \end{aligned}$$

Therefore, the new response time is no smaller than the original one.

- Suppose that at $r_{k,s}$ and just before $r_{k,s}$ there are at least m higher priority tasks active. Let A be the latest time before $r_{k,s}$ such that there are no more than $m-1$ higher priority active tasks at $A-1$. By setting $r'_{k,s} = A$, we observe that task τ_k cannot execute in $[A, r_{k,s}]$, because all processors are busy executing higher priority tasks. Also, after $r_{k,s}$ the schedule does not change, and particularly $f'_{k,h} = f_{k,h}$. So the new response time of $J_{k,h}$ is

$$\begin{aligned} R'_{k,h} &= f'_{k,h} - r'_{k,h} \\ &= f_{k,h} - (r_{k,h} - (r_{k,s} - A)) \\ &= R_{k,h} + (r_{k,s} - A) > R_{k,h} \end{aligned}$$

  i.e., the new response time is strictly larger than the original one.

□

7.1.1 Pessimism and optimism in RTA-LC

The original form of RTA-LC is presented in Section 6.4.2. Here we point out both the pessimism and optimism in it.

Pessimism in the iteration procedure

Let us first consider the following task set running on two processors: $\tau_1 = (28, 50, 50)$, $\tau_2 = (13, 30, 30)$, $\tau_3 = (5, 50, 50)$, $\tau_4 = (6, 30, 30)$, and $\tau_5 = (6, 40, 40)$. Following Theorem 18, we compute the WCRT upper bound of first four tasks and obtain: $R_1 = 28$, $R_2 = 13$, $R_3 = 18$ and $R_4 = 24$. While iteratively computing the WCRT of task τ_5 and along with x increasing, the CI task set that corresponds to the maximised $\Omega_5(x)$ varies. For example, when $x = 31$, the total interference $\Omega_5(x)$ is maximised with τ_4 as the CI task; and when $x = 38$, $\Omega_5(x)$ is maximised if τ_3 is the CI task. Finally, the iteration in Theorem 18 will report a deadline miss for τ_5. However, this task set is indeed schedulable, according to our improved analysis method that will be introduced in next section.

We now formalize the pessimism in the iteration procedure of RTA-LC. For simplicity we focus on the constrained-deadline case, but the pessimism also exists in the unconstrained-deadline case. Let $\mathcal{T}^{CI_1}$ and $\mathcal{T}^{CI_2}$ be two possible candidates of the carry-in task set. Suppose $\mathcal{T}^{CI_1}$ is indeed the carry-in task set that leads to the worst-case response time of the analysed task τ_k, then during the whole analysis procedure the total interference to task τ_k can be bounded by

$$\Omega_k^1(x) = \sum_{\tau_i \notin \mathcal{T}^{CI_1}} I_k^{NC}(\tau_i, x) + \sum_{\tau_i \in \mathcal{T}^{CI_1}} I_k^{CI}(\tau_i, x)$$

and suppose with the above calculation of $\Omega_k^1(x)$ the fixed-point iteration procedure converges at x_1.

Similarly, if $\mathcal{T}^{CI_2}$ is indeed the carry-in task set that leads to the worst-case response time of the analysed task τ_k, then during the whole analysis procedure the total interference to task τ_k can be bounded by

$$\Omega_k^2(x) = \sum_{\tau_i \notin \mathcal{T}^{CI_2}} I_k^{NC}(\tau_i, x) + \sum_{\tau_i \in \mathcal{T}^{CI_2}} I_k^{CI}(\tau_i, x)$$

and the fixed-point iteration procedure converges at x_2.

In general, $\Omega_k^1(x)$ and $\Omega_k^2(x)$ may not dominate each other. In other words, we may have $\Omega_k^1(x') > \Omega_k^2(x')$ for some x' and $\Omega_k^1(x'') < \Omega_k^2(x'')$ for some x''. In particular, we assume it holds

$$\Omega_k^1(x_1) < \Omega_k^2(x_1)$$
$$\Omega_k^2(x_2) < \Omega_k^1(x_2)$$

Since the interference upper bound $\Omega_k(x)$ calculated by Equation (6.10) upper bounds both Ω_k^1 and Ω_k^2, we know

$$\Omega_k^1(x_1) < \Omega_k(x_1)$$
$$\Omega_k^2(x_2) < \Omega_k(x_2)$$

Therefore, we can conclude that in RTA-LC (using Ω_k calculated by Equation (6.10)), the fixed-point iteration may not converge at either x_1 or x_2, but converges at some point which is larger than both x_1 and x_2.

Pessimism in the carry-in workload of constrained-deadline tasks

Given a task $\tau_i = (19, 50, 50)$ with $R_i = 20$ and $x = 29$. By using Lemma 6, there is $W_i^{CI}(x) = 19$. However, since we know τ_i is running before the problem window, we can observe that the workload of τ_i in a window of length 29 cannot exceed 18. As the minimum time interval between a job's finishing point and the successive job's arrival for τ_k is $T_i - R_i = 30$.

Optimism in the carry-in workload of unconstrained-deadline tasks

Since we allow $D_i > T_i$, there can be $R_i > T_i$. Suppose a task τ_i is a CI task and $\left\lceil \frac{R_i}{T_i} \right\rceil = 3$. This means at the same moment, there can be at most 3 active jobs from τ_i in the system. And given a problem window with length equivalent to T_i, the workload upper bound of τ_i should be $\geq 3 \cdot C_i - 1$. But the computation result returned by Lemma 6 is $2 \cdot C_i - 1$.

7.2 RTA-CE: RTA with Carry-in Enumeration

In this section we introduce a new response time analysis that solves the problems described above.

7.2.1 New workload upper bound

We start by proposing a new upper bound on the workload.

Lemma 9. *The workload bound of a carry-in task τ_i in the problem window of length x can be computed as:*

$$W_i^{CI}(x) = W_i^{NC}([x - x_p]_0) + [x]^\delta \tag{7.1}$$

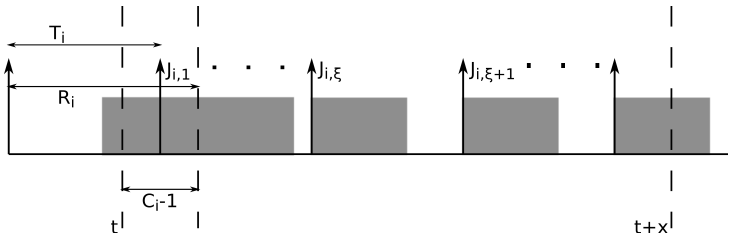


Figure 7.1: The CI workload of a task.

where x_p and δ are defined as:

$$x_p = C_i - 1 + \left\lceil \frac{R_i - C_i}{T_i - C_i} \right\rceil T_i - R_i \tag{7.2}$$

$$\delta = \left\lceil \frac{R_i - C_i}{T_i - C_i} \right\rceil C_i - 1 \tag{7.3}$$

and W^{NC} is calculated as in Equation (6.6)*.*

Proof. We use w to denote the number of carry-in jobs of τ_i. Suppose t is the starting time of the problem window, then by the definition of critical instant, there are at most $m-1$ tasks having active jobs at time $t-1$, so the carry-in jobs are already executing at $t-1$, and the total unfinished execution demand of the carry-in jobs by the start of the problem window is bounded by $w \cdot C_i - 1$.

We argue that the worst-case release pattern leading to the maximal workload of τ_i is as follows: (1) the earliest carry-in job executes as late as possible and finishes at exactly its worst-case response time, (2) the earliest carry-in job starts execution one time unit before the starting point of the problem window, (3) inside the problem window all jobs are released as fast as possible. The argument follows the same strategy as in [Bak03, GSYY09], showing that if we shift the release times in any way, this will not increase the total workload of the job inside the problem window. We omit the detailed reasoning procedure, but we use Figure 7.1 to depict this worst-case scenario. In the following, we show that with this particular release pattern, the workload of τ_i in the problem window is bounded by Equation (7.1).

We index the job released inside problem window as $J_{i,1}, J_{i,2}, \ldots$, and we assume ξ be the smallest index (if there exists) such that, when $J_{i,\xi}$ is released, all the preceding jobs have been finished. Therefore, the time interval between the starting time of the problem window and the release time of $J_{i,\xi}$ must be at least enough to execute all its preceding jobs. On the other hand, the release time of the ξ^{th} job in the problem window is $x_p = (C_i - 1) + (w + \xi - 1) \cdot T_i - R_i$. So we have

$$w \cdot C_i - 1 + (\xi - 1)C_i \leq (C_i - 1) + (w + \xi - 1) \cdot T_i - R_i$$

Since ξ is an integer, the minimum value for it is:

$$\xi = \left\lceil \frac{R_i - C_i}{T_i - C_i} \right\rceil - (w - 1)$$

Then we can obtain the expression of x_p as in Equation (7.2).

The total workload of τ_i in the problem window is contributed by two parts: (i) the jobs executing *before* the release of the ξ^{th} job; (ii) the jobs executing *after* the release of the ξ^{th} job.

The jobs executing before the ξ^{th} job include the carry-in jobs (the workload of which in the problem window is bounded by $w \cdot C_i - 1$ as we discussed above), and the $\xi - 1$ jobs that are released in the

problem window. So their total workload is bounded by

$$\delta = w \cdot C_i - 1 + (\xi - 1)C_i = \left\lceil \frac{R_i - C_i}{T_i - C_i} \right\rceil C_i - 1$$

On the other hand, the maximal workload actually executed in the problem window is bounded by the length of the problem window x.

When τ_i releases a job at x_p, all its preceding jobs have been finished, so the total workload of jobs released in $[x_p, x]$ is bounded by

$$W_i^{NC}([x - x_p]_0)$$

Putting the workload upper bound of the two parts together establishes the lemma. □

Our new formulation in Equation (7.1) for calculating $W_i^{CI}(x)$ fixes both the pessimism and optimism issues in Lemma 6. Moreover, our new calculation of $W_i^{CI}(x)$ leads to an interesting fact: now, when $R_i < T_i$, the two functions $W_i^{CI}(x)$ and $W_i^{NC}(x)$ are in general incomparable. More specifically, $W_i^{CI}(x)$ may be smaller than $W_i^{NC}(x)$ with certain x. This can be demonstrated by the following example. Consider the analysis of the workload of task τ_i with $C_i = 2$, $T_i = 4$ and $R_i = 3$, whose workload function is depicted in Figure 7.2. We have $W_i^{CI}(2) = 1 < W_i^{NC}(2) = 2$. This (somehow counter-intuitive) phenomenon is due to the fact that a carry-in job has at least been executed for one time unit before the start of the problem window. However, when $R_i \geq T_i$, the relation $W_i^{CI}(x) \geq W_i^{NC}(x)$ still holds.

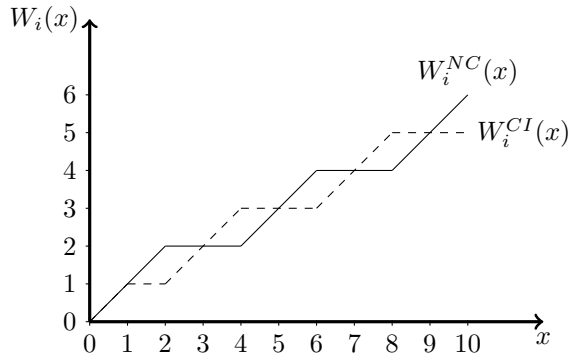


Figure 7.2: The worst-case workload of a task with $C_i = 2$, $T_i = 4$ and $R_i = 3$

Finally, we can still compute the carry-in and non-carry-in interference $I_k^{CI}(\tau_i, x, h)$ and $I_k^{NC}(\tau_i, x, h)$ of a higher priority task τ_i to the analysed task τ_k based on $W_i^{CI}(x)$ and $W_i^{NC}(x)$ by Equation (6.8) and Equation (6.9).

7.2.2 New iterative analysis procedure

Given a specific carry-in task set $\mathcal{T}^{CI}$, we define the total interference (over time interval x) on task τ_k as follows :

$$\Omega_k(\mathcal{T}^{CI}, x, h) = \sum_{\tau_i \notin \mathcal{T}^{CI}} I_k^{NC}(\tau_i, x, h) + \sum_{\tau_i \in \mathcal{T}^{CI}} I_k^{CI}(\tau_i, x, h) \tag{7.4}$$

We denote $\mathcal{X}^{h,\mathcal{T}^{CI}}$ the minimal solution of the following iteration :

$$x = \left\lfloor \frac{\Omega_k(\mathcal{T}^{CI}, x, h)}{m} \right\rfloor + h \cdot C_k \tag{7.5}$$

Lemma 10. *Given a task τ_k and a carry-in task set $\mathcal{T}^{CI}$, the WCRT of τ_k with respect to $\mathcal{T}^{CI}$ is upper bounded by :*

$$R_k^{\mathcal{T}^{CI}} = \max_{h \in [1,H]} \{\mathcal{X}^{h,\mathcal{T}^{CI}} - (h-1) \cdot T_k\} \tag{7.6}$$

Proof. We show here the case when deadlines are less than or equal to periods and the arbitrary deadline case can be derived similarly. For simplicity, we use $I_k(\tau_i, x)$, $\Omega_k(\mathcal{T}^{CI}, x)$ and $\mathcal{X}^{\mathcal{T}^{CI}}$ by ignoring $h = 1$.

Suppose $\mathcal{X}^{\mathcal{T}^{CI}}$ is the minimal solution of the iteration

$$x = \left\lfloor \frac{\Omega_k(\mathcal{T}^{CI}, x)}{m} \right\rfloor + C_k \tag{7.7}$$

We are going to prove $\mathcal{X}^{\mathcal{T}^{CI}}$ is an upper bound of τ_k's response time with respect to $\mathcal{T}^{CI}$.

The proof is similar as in [BC07b]. By contradiction. Suppose, for some carry-in task set $\mathcal{T}^{CI}$, the iteration in Equation (7.7) ends with a value $\mathcal{X}^{\mathcal{T}^{CI}} \leq D_k$, but the response time of τ_k, released with the same $\mathcal{T}^{CI}$, is higher than $\mathcal{X}^{\mathcal{T}^{CI}}$. Since the iteration ends, there is

$$\mathcal{X}^{\mathcal{T}^{CI}} = \left\lfloor \frac{\Omega_k(\mathcal{T}^{CI}, \mathcal{X}^{\mathcal{T}^{CI}})}{m} \right\rfloor + C_k$$

That is,

$$\mathcal{X}^{\mathcal{T}^{CI}} = \left\lfloor \frac{\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}})}{m} \right\rfloor + C_k \tag{7.8}$$

Remind that $\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}})$ is a short form for

$$\sum_{\tau_i \notin \mathcal{T}^{CI}} I_k^{NC}(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}}) + \sum_{\tau_i \in \mathcal{T}^{CI}} I_k^{CI}(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}})$$

According to Theorem 3 in [BC07b], if R_k^{ub} is a response time upper bound of τ_k, we have

$$\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, R_k^{ub}) < m(R_k^{ub} - C_k + 1)$$

We assumed that $\mathcal{X}^{\mathcal{T}^{CI}}$ is not an upper bound, hence we reverse the above formula:

$$\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}}) \geq m(\mathcal{X}^{\mathcal{T}^{CI}} - C_k + 1) \tag{7.9}$$

Then, by replacing $\sum_{\tau_i \in \mathcal{T}} I_k(\tau_i, \mathcal{X}^{\mathcal{T}^{CI}})$ in Equation (7.8) with the right hand side of Equation (7.9), we have

$$\begin{aligned} \mathcal{X}^{\mathcal{T}^{CI}} &\geq (\mathcal{X}^{\mathcal{T}^{CI}} - C_k + 1) + C_k \\ &= \mathcal{X}^{\mathcal{T}^{CI}} + 1 \end{aligned}$$

By contradiction, we know that $\mathcal{X}^{\mathcal{T}^{CI}}$ is the upper bound of response time of τ_k with carry-in task set $\mathcal{T}^{CI}$. □

Theorem 21 (RTA-CE)**.** *The upper bound of WCRT of task τ_k is*

$$R_k = \max_{\mathcal{T}^{CI} \in \mathcal{Z}} \left(R_k^{\mathcal{T}^{CI}} \right) \tag{7.10}$$

with $\mathcal{Z}$ being the set of all possible CI task sets.

Proof. As long as τ_k is released at one of its critical instants, its response time is then upper bounded by R_k, which, according to Theorem 20, is the upper bound of τ_k's worst-case response time. □

Note that with our new formulation of computing $W_i^{CI}(x)$ the overall analysis procedure still guarantees to terminate under the condition as in Equation (6.13). This follows the similar reasoning in [GSYY09].

The WCRT upper bound returned by RTA-CE will never be larger than the result of RTA-LC. As the cost of a more precise response time estimation, the new RTA needs to explicitly enumerate all possible carry-in task sets in $\mathcal{Z}$ (this is where the name RTA-CE comes from) and this leads to an exponential time complexity for the new RTA. However, as shown later in the simulation, the RTA-CE can handle task systems with realistic scales in reasonable time.

7.2.3 Improving the efficiency

The iterative fixed-point calculation in RTA-CE with each given h should start with an initial value no larger than the minimal solution of Equation (7.5) (otherwise the iteration converges at a larger solution and results in more pessimistic response time bounds). Under this constraint, the initial value should be as large as possible, to make the iteration procedure converge faster. A straightforward and safe initial value is $x = h \cdot C_k$. In the following, we introduce a larger safe initial value to speedup the analysis procedure. We first define:

$$\omega_k(x,h) = \sum_{\tau_i \in \mathcal{T}} \min\{I_k^{NC}(\tau_i, x, h), I_k^{CI}(\tau_i, x, h)\} \tag{7.11}$$

Note that, as we discussed in Section 7.2.1, W_i^{NC} and W_i^{CI} are generally incomparable, and so do I_k^{NC} and I_k^{CI}. Let s^h be the minimal solution of the following iteration starting with $x = h \cdot C_k$:

$$x = \left\lfloor \frac{\omega_k(x,h)}{m} \right\rfloor + h \cdot C_k \tag{7.12}$$

For any $\mathcal{T}^{CI}$, it holds $\omega_k(x,h) \leq \Omega_k(\mathcal{T}^{CI}, x, h)$, so we know s^h is no larger than $\mathcal{X}^{h,\mathcal{T}^{CI}}$ for any $\mathcal{T}^{CI}$. Therefore, s^h can be used as a safe initial value for x in Equation (7.5) for any $\mathcal{T}^{CI}$.

Furthermore, to find $\mathcal{X}^{h,\mathcal{T}^{CI}}$, we can utilise the already computed $\mathcal{X}^{h-1,\mathcal{T}^{CI}}$ and start with $x = C_k + \mathcal{X}^{h-1,\mathcal{T}^{CI}}$.

In conclusion, the starting point for Equation (7.5) is

$$x = \begin{cases} s^h & h = 1 \\ \max\{s^h, C_k + \mathcal{X}^{h-1,\mathcal{T}^{CI}}\} & h > 1 \end{cases} \tag{7.13}$$

7.3 Evaluation

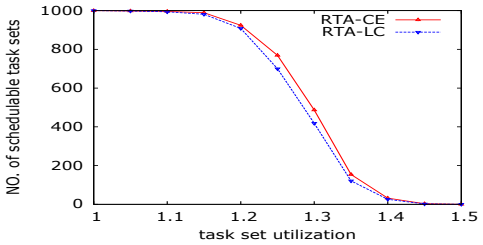
Each test in the evaluation is described by a tuple (m, n, U) where m is the number of processors, n is the number of tasks and U is the total task utilisation in the task set. Task sets are randomly generated according to Randfixedsum algorithm in [ESD10]. Every task has its period T_i uniformly distributed in the range $[100, 200]$. For constrained-deadline tasks, the ratio between D_i and T_i is distributed in the range $[0.7, 1]$; for arbitrary-deadline tasks, $\frac{D_i}{T_i} \in [0.7, 1.3]$. Before applying RTA-CE and RTA-LC, tasks in a task set are first assigned priorities by the Deadline Monotonic (DM) strategy, i.e. a task with shorter deadline gets a higher priority.

7.3.1 Performance tests

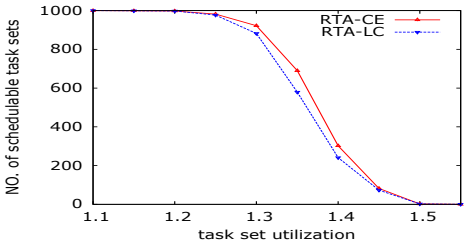
We conduct simulations with randomly generated task sets to evaluate the precision improvement of RTA-CE over RTA-LC. We consider $m \in \{2, 4\}$, and given m we set its corresponding task set size n and task set utilisation U to be $n = 10 \cdot m$ and $U \in \{0.025m, 0.5m, \ldots, 0.975m, m\}$, respectively. For each configuration (m, n, U), 1000 task sets are randomly generated.

We report the results in Figure 7.3. The horizontal axis specifies task set utilisations and the vertical axis denotes the number of schedulable systems found. We omit plotting points where the number of schedulable task sets is simply 1000 or 0 in figures.

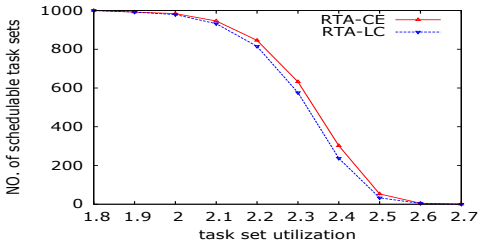
The simulation results confirm RTA-CE's improvement over RTA-LC in practice. For example, in the case of tasks with arbitrary deadlines where $m = 2$, $n = 20$ and $U = 1.35$, RTA-CE finds 111 more schedulable task sets (among 1000 randomly generated task sets) than RTA-LC.



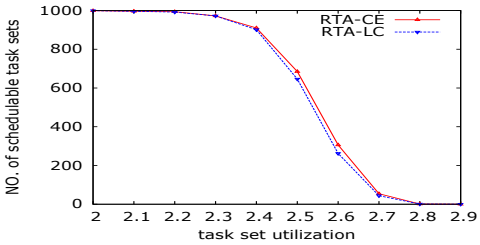
(a) $m = 2, n = 20, \frac{D_i}{T_i} \in [0.7, 1]$



(b) $m = 2, n = 20, \frac{D_i}{T_i} \in [0.7, 1.3]$

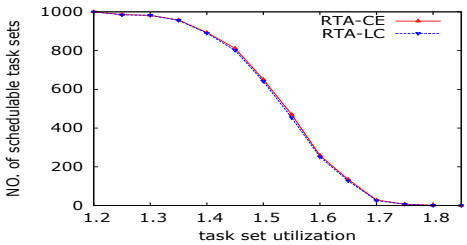


(c) $m = 4, n = 40, \frac{D_i}{T_i} \in [0.7, 1]$

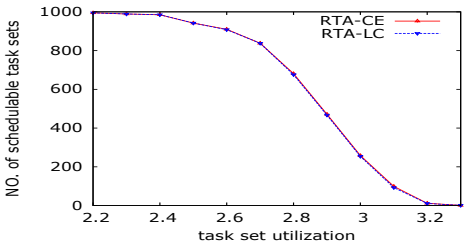


(d) $m = 4, n = 40, \frac{D_i}{T_i} \in [0.7, 1.3]$

Figure 7.3: RTA-CE v.s. RTA-LC



(a) $m = 2, n = 20$



(b) $m = 4, n = 40$

Figure 7.4: RTA-CE v.s. RTA-LC ($T_i \in [10, 1000]$, $\frac{D_i}{T_i} \in [0.7, 2]$)

The advantage of RTA-CE over RTA-LC may not be always as shown in Figure 7.3. For example, if we assume task periods are distributed in range $[10, 1000]$ and $\frac{D_i}{T_i} \in [0.7, 2]$, we obtain the simulation

results in Figure 7.4. On the other hand, it is possible to set up different simulations and get even stronger dominance relation between RTA-CE and RTA-LC than Figure 7.3 shows.

7.3.2 Efficiency tests

The efficiency tests contain two parts:

- We investigate how much the efficiency improvement scheme introduced in Section 7.2.3 can affect the run-time performance of RTA-CE;
- we demonstrate to which extent RTA-CE can (or cannot) scale through a showcase study.

All tests are conducted in a MacBook with 2.5 GHz Intel Core i5 processor.

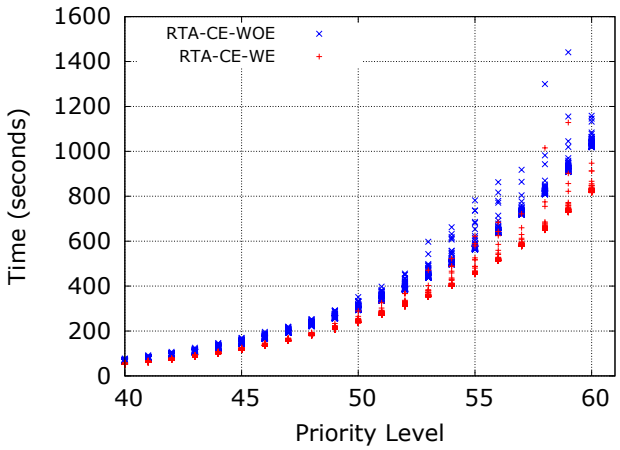


Figure 7.5: Efficiency improvement tests

At first, we discuss the simulation for testing RTA-CE's efficiency strategy. To set up tests, we consider $m = 6$, $n = 60$, $U \in \{3, 4, 5, 6\}$ and arbitrary-deadline tasks. For every (m, n, U), 30 task sets are randomly generated. In the simulation, we measure the time spent for deciding the schedulability of tasks by RTA-CE with efficiency enhancement (RTA-CE-WE) and without efficiency enhancement (RTA-CE-WOE).

There are 60 different priority levels (1-60) in each generated task set and a lower number means higher priority. RTA procedure is applied from highest priority task (with priority 1) to lowest priority task (with priority 60). For all task sets we report the accumulated time, which starts from applying RTA-CE on the task with priority 1, that is used by RTA-CE-WE and RTA-CE-WOE respectively to decide the schedulability at each priority level. In this way, for a schedulable task set, the time obtained for priority level 60 is actually the time used to decide the schedulability of a task set. For a task set that is not schedulable, we report time instances recorded till the last task that RTA-CE checks.

The final results are plotted in Figure 7.5. On average, RTA-CE-WE saves $1/5 \sim 1/4$ RTA-CE-WOE's run-time. For instance, the time RTA-CE-WE uses to deal with 60 tasks and the time RTA-CE-WOE spends on 58 tasks are almost the same. And RTA-CE-WE can always obtain the analysis result (schedulable or unschedulable) for a task set in less than 20 minutes.

To further stress the efficiency improvements, we randomly generate an arbitrary-deadline task set with $m = 8$, $n = 80$ and $U = 4$. Then we measured the run-time for applying RTA-CE on this task set.

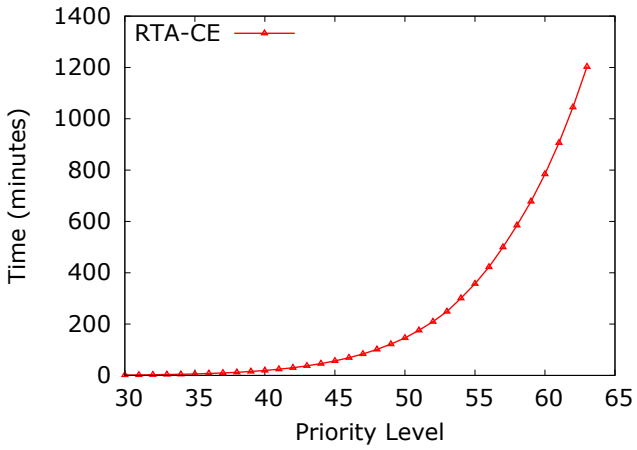


Figure 7.6: The scalability test ($m = 8, n = 80, U = 4$)

The test result is reported in Figure 7.6. We manually stopped the test after the run-time exceeded 20 hours: at that instant, RTA-CE had finished the schedulability check of the first 63 tasks.

7.4 Conclusion

In this chapter we considered the response time analysis problem for global fixed-priority scheduling on multiprocessors. We proved the existence of a type of critical instant leading to the worst-case response time of a task. The idea of this critical instant has been used in the context of approximated analysis, but has not been strictly proved to lead to the actual worst-case response time for arbitrary-deadline task sets.

Then we improved the state-of-the-art technique RTA-LC by addressing both its pessimism and optimism. We first propose a new formula to bound the workload of carry-in jobs. The new formula is, on one hand more precise than the one used in RTA-LC, and on the other hand it fixes the potential under-estimation of the carry-in workload for unconstrained-deadline tasks. We then proposed a new iterative response time analysis procedure that achieves better precision. Simulations with randomly generated tasks show that our new method RTA-CE can successfully accept a considerable number of task sets that are deemed to be unschedulable by RTA-LC.

Chapter 8

New Techniques for G-EDF Schedulability Analysis

In this chapter, we address the problem of schedulability analysis for a set of sporadic real-time tasks (with constrained deadlines) scheduled by the Global Earliest Deadline First (G-EDF) policy on a multiprocessor platform. When it comes to multiprocessor global schedulability analysis, state-of-the-art tests are often incomparable. That is, a task set that is judged not schedulable by a test may be verified to be schedulable by another test, and vice versa.

We first develop a new schedulability test that integrates the limited carry-in technique and Response Time Analysis (RTA) procedure for G-EDF schedulability analysis. Then, we provide an over-approximate variant of this test with better run-time efficiency. Later, we extend these two tests to self-suspending tasks. All schedulability tests proposed in this chapter have provable dominance over their state-of-the-art counterparts.

Finally, we conduct extensive comparisons among different schedulability tests. Our new tests bring a significant improvement for schedulability analysis for G-EDF.

The content in this chapter is based on [SL15].

8.1 An Improved Schedulability Test for G-EDF

In Section 6.3, we introduced two state-of-the-art tests for G-EDF scheduling: BC shows us the classic RTA procedure is still effective in multiprocessor schedulability analysis; Bar contributes the idea that we may more precisely estimate the interference on multiprocessor global scheduling by limiting the number of CI tasks. However, all tasks in BC are considered as CI tasks, whereas Bar directly measures the interference in a time interval instead of using the iterative analysis. Therefore, here we combine the two techniques to take advantage of their strong points.

Before proceeding with the presentation of the new algorithm, we quickly recapitulate the related concepts; we further propose the definition of a *sub problem window*, as depicted in Figure 8.1.

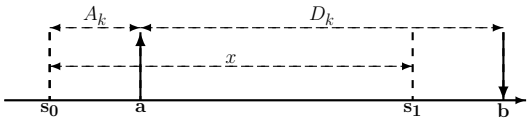


Figure 8.1: Different problem windows

We are checking the schedulability of a target task τ_k, so we select a *target job* of τ_k in the window

$[a, b)$ with a and b corresponding to its release time and deadline, respectively. We still use the extended problem window $[s_0, b)$, where s_0 is the earliest time point before a such that within $[s_0, a)$ all processors are busily executing jobs with absolute deadlines smaller than or equal to the target job's and $A_k = a - s_0$ is the length of this busy period.

In the following, instead of simply computing the interference generated by a task in the extended problem window $[s_0, b)$, we follow the iterative technique in BC to calculate the interference. We define a new time interval $[s_0, s_1)$, called *sub problem window of the extended one*, such that $s_0 < s_1 \leq b$, and we would like to compute the interference in the sub problem window $[s_0, s_1)$. We denote $x = s_1 - s_0$.

8.1.1 Interference in a sub problem window

Before upper bounding the interference in the sub problem window $[s_0, s_1)$, we first formulate the workload, subject to G-EDF scheduling, generated by a task τ_i in this sub problem window, denoted as $W_i(x, \mathcal{L})$ with $\mathcal{L} = A_k + D_k$. We use $W_i^{NC}(x, \mathcal{L})$ for NC tasks and $W_i^{CI}(x, \mathcal{L})$ for CI tasks, and $W_i^{DIFF}(x, \mathcal{L}) = W_i^{CI}(x, \mathcal{L}) - W_i^{NC}(x, \mathcal{L})$.

Algorithm 3 calculates the maximum workload produced by a NC task inside $[s_0, s_1)$. It follows the worst-case release pattern for $W_i^{NC}(x, \mathcal{L})$ such that

- the first job (carry-in job) of τ_i is released at time s_0;
- successive jobs of τ_i are released as soon as possible.

Algorithm 3: $W_i^{NC}(x, \mathcal{L})$

```
1 W ← 0, p ← 0
2 while p < x do
3     if p + D_i ≤ L then
4         W ← W + [[x − p]]^{C_i}
5         p ← p + T_i
6     else
7         break
8 return W
```

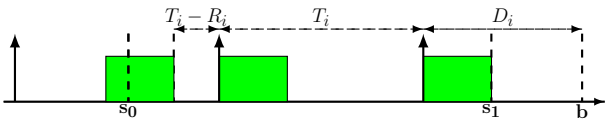


Figure 8.2: The worst-case arrival pattern for $W_i^{CI}(x, \mathcal{L})$

As for a CI task τ_i, its worst-case release pattern for $W_i^{CI}(x, \mathcal{L})$ corresponds to the scenario in Figure 8.2:

- the carry-in job finishes its execution with τ_i's worst-case response time;
- successive jobs of τ_i are released as soon as possible;
- the carry-out job arrives as late as possible and finishes its worst-case execution within the sub window.

It is easy to see that any other scenario produces a workload that is not greater than this one. Note that all interfering jobs, including the carry-out one, have their absolute deadlines no later than b.

Algorithm 4 uses this worst-case release pattern to compute $W_i^{CI}(x, \mathcal{L})$. At first, it computes the latest possible arrival time for the carry-out job, denoted as $[\![x - C_i]\!]^{\mathcal{L}-D_i}$. If the resulting arrival time is less than 0, it means that there is at most one interfering job of τ_i in the (sub) window and it must arrive before the beginning of the busy period; then lines 3-6 in the algorithm deal with such a situation. In the other case, there will be $(N+1)$ interfering jobs of τ_i that can contribute their complete worst-case executions; and the remaining part (lines 7-9) of the algorithm deals with this situation.

Algorithm 4: $W_i^{CI}(x, \mathcal{L})$

```
1 W ← 0
2 p ← [[x − C_i]]^{L−D_i}
3 if p < 0 then
4 |   W ← [[L − (D_i − R_i)]]^{C_i}
5 |   W ← [[W]]_0^x
6 |   return W
7 N ← ⌊p / T_i⌋
8 W ← (N + 1) · C_i + [[p mod T_i − (T_i − R_i)]]_0^{C_i}
9 return W
```

For τ_k, we only need to consider its job releases before the target one. Then, its NC and CI workload can be further bounded.

$$W_k^{NC}(x, \mathcal{L}) = \left[\!\left[W_k^{NC}(x, \mathcal{L})\right]\!\right]^{I_k^{NC}([\![\mathcal{L}-T_i]\!]_0)}$$
$$W_k^{CI}(x, \mathcal{L}) = \left[\!\left[W_k^{CI}(x, \mathcal{L})\right]\!\right]^{I_k^{CI}([\![\mathcal{L}-T_i]\!]_0)}$$

After computing the workload, the interference can be bounded. The corresponding NC interference and CI interference by τ_i on the target job of τ_k are denoted as $I_{i,k}^{NC}(x, A_k)$ and $I_{i,k}^{CI}(x, A_k)$. For any $x \geq A_k + C_k$, the interference from τ_i is bounded as follows.

$$I_{i,k}^{NC}(x, A_k) = \left[\!\left[W_i^{NC}(x, \mathcal{L})\right]\!\right]^{x-C_k+1}$$
$$I_{i,k}^{CI}(x, A_k) = \left[\!\left[W_i^{CI}(x, \mathcal{L})\right]\!\right]^{x-C_k+1}$$

$I_{i,k}^{DIFF}(x, A_k)$ is defined as $(I_{i,k}^{CI}(x, A_k) - I_{i,k}^{NC}(x, A_k))$.

Given the sub window $[s_0, s_1)$, we now demonstrate how to upper bound the total interference on the target job.

- According to the limited carry-in technique in Bar, we can derive an upper bound to the total interference, denoted as Ω_1 such that

$$\Omega_1 = \sum_{\tau_i \in \mathcal{T}} I_{i,k}^{NC}(x, A_k) + \max_{\text{the } m-1 \text{ largest}} I_{i,k}^{DIFF}(x, A_k) \tag{8.1}$$

- On the other hand, given that $[s_0, a)$ is a busy period with length A_k, we can formulate another upper bound Ω_2 such that

$$\Omega_2 = m \cdot A_k + \sum_{i \neq k} I_{i,k}^{CI}(x - A_k, 0) \tag{8.2}$$

  where $\sum_{i \neq k} I_{i,k}^{CI}(x - A_k, 0)$ upper bounds the maximum interference in interval $[a, s_1)$ by assuming all tasks different from τ_k as CI tasks.

- Finally, we define the total interference upper bound in the sub window $[s_0, s_1)$ as $\Omega(x, A_k)$, which

is the smaller one between Ω_1 and Ω_2; that is

$$\Omega(x, A_k) = [\![\Omega_1]\!]^{\Omega_2} \tag{8.3}$$

8.1.2 RTA-LC-EDF

Now, we are going to formulate a new RTA procedure for G-EDF schedulability analysis that integrates the limited carry-in technique, and we call the new test "RTA with Limited Carry-in for multiprocessor global EDF scheduling": RTA-LC-EDF.

Theorem 22 (RTA-LC-EDF)**.** *Given an A_k value, let us say X_{A_k} be solution of the following iteration starting from $X = A_k + \phi$ with $\phi = C_k$.*

$$X \leftarrow \phi + \left\lfloor \frac{\Omega(X, A_k)}{m} \right\rfloor \tag{8.4}$$

Then, the response time upper bound of τ_k is

$$R_k = \max_{\forall A_k}\{X_{A_k} - A_k\}.$$

Proof. The theorem can be proved by showing that, for any A_k such that within the time interval $[s_0, a)$ all processors are fully occupied by higher priority jobs, X_{A_k} is an upper bound of the target job's finishing time (let us say $s_0 = 0$) subject to this specific A_k value.

Suppose Equation (8.4) converges to X_{A_k}. If X_{A_k} is not an upper bound of τ_k's finishing time, then there would be $\lfloor \frac{\Omega(X_{A_k}, A_k)}{m} \rfloor + \phi > X_{A_k}$, which contradicts with Equation (8.4)'s convergence at X_{A_k}. □

Then, the same refinement procedure as in BC can be applied to RTA-LC-EDF.

1. We start by setting $R_i = D_i$ for every task and by marking all tasks as "potentially unschedulable".
2. We compute R_i for every task by using the iterative procedure in Theorem 22. For a potentially unschedulable task, if the resulting response time is $\leq D_i$, then it is marked as "schedulable" and the value of R_i is updated; for a schedulable task, if the resulting response time $< R_i$, then R_i is updated.
3. If no task has R_i update, the iteration stops; otherwise, go back to step (2).

In the following, we are going to prove that the new test dominates Bar and BC: if a task set is decided schedulable by Bar or BC, the same result is returned by RTA-LC-EDF.

Theorem 23. *RTA-LC-EDF $\succeq$ Bar and RTA-LC-EDF $\succeq$ BC.*

Proof. Here we give the key points to prove RTA-LC-EDF's dominance over Bar.

- For any A_k and for any $x \in [A_k + C_k, A_k + D_k]$, there is $\Omega \geq \Omega(x, A_k)$. Remember that Ω (Equation (6.4)) is the total interference, with respect to corresponding A_k value, computed by Bar in the extended problem window; and $\Omega(x, A_k)$ is the total interference computed by RTA-LC-EDF in the sub problem window with a length x.
- For any $A_k > \overline{A}_k$ (Equation (6.5)), there is $\Omega \leq m(A_k + D_k - C_k)$ (please check [Bar07] for more details), this implies $\Omega(x, A_k) \leq m(A_k + D_k - C_k)$.

In the end, if Equation (6.3) in Bar holds (i.e. $\Omega \leq m(A_k + D_k - C_k)$ and τ_k is schedulable), then there will be $\Omega(x, A_k) \leq m(A_k + D_k - C_k)$ for any $x \in [A_k + C_k, A_k + D_k]$; and there is no way for RTA-LC-EDF to result in a response time upper bound larger than D_k.

The dominance of RTA-LC-EDF over BC is rather straightforward. For any $x \geq A_k + C_k$, it is easy to see that $I_{i,k}(x - A_k) \geq I^{CI}_{i,k}(x - A_k, 0)$, where $I_{i,k}(x)$ is the interference estimation by BC. Thus, $\Omega(x, A_k) \leq \Omega_2 \leq \sum_{i \neq k} I_{i,k}(x - A_k) + mA_k$. In the end, RTA-LC-EDF will return a WCRT upper bound no larger than the one returned by BC. □

8.1.3 Upper bound to A_k

The RTA-LC-EDF test in Theorem 22 does not provide a bounded range of A_k values for the analysis. In order to apply the new test, we must find a finite set of A_k such that it is enough to estimate the response time upper bound by simply using these A_k values. This is what we are going to present in this section. We remind that A_k denotes the length of the busy period (before the release of target job).

As a first step, we restrict to valid A_k's values for RTA-LC-EDF, where the concept of a valid busy period length is defined below.

Definition 14 (Valid A_k)**.** *Given a busy period, we say it has a valid length A_k if the solution of following iteration would be larger than A_k.*

$$X \leftarrow \left\lfloor \frac{W(X)}{m} \right\rfloor \tag{8.5}$$

with $W(X) =$

$$\sum_{\tau_i \in \mathcal{T}} W^{NC}_i(X, \mathcal{L}) + \max_{\textit{the } m-1 \textit{ largest}} W^{DIFF}_i(X, \mathcal{L}). \tag{8.6}$$

The $W(X)$ formulated above represents the total workload in a sub problem window with length X. A busy period with a valid A_k means that all processors are always occupied in the interval $[s_0, s_0 + A_k)$ by higher priority jobs and the resulting workload must cause an interference on the execution of target job. Thus, in the analysis, we exclude those busy period lengths such that the solution of Equation (8.5) is no larger than its corresponding A_k. In the special case that there is no valid A_k, it is safe to conclude that the WCRT of the target task τ_k is C_k.

Then, we explore the two following properties to find an upper bound to A_k.

Theorem 24. *It is sufficient to bound the busy period length A_k by $A_k < A^{\alpha}_k$ such that*

$$A^{\alpha}_k = \frac{C_\Sigma + \sum_{\tau_i \in \mathcal{T}} (T_i - C_i)U_i}{m - U_{tot}}.$$

Proof. We first formulate a *generic NC* workload function for a task τ_i in a time interval t without restricting to specific scheduling policy.

$$w_i(t) = \left\lfloor \frac{t}{T_i} \right\rfloor C_i + [\![t \bmod T_i]\!]^{C_i}$$

A linearised upper bound for $w_i(t)$ is the following:

$$lw_i(t) = U_i t + (T_i - C_i)U_i$$

It can be easily seen that $W^{NC}_i(A_k, A_k + D_k) \leq lw_i(A_k)$, and $W^{CI}_i(A_k, A_k + D_k) \leq lw_i(A_k) + C_i$. Thus, in order to have a busy period with valid length A_k, a necessary condition is that

$$C_\Sigma + \sum_{\tau_i \in \mathcal{T}} lw_i(A_k) > m \cdot A_k$$

In the end, we have $A_k < A_k^\alpha$. □

Theorem 25. *It is sufficient to bound the busy period length A_k by $A_k < A_k^\beta$ such that*

$$A_k^\beta = \frac{C_\Sigma + \sum_{\tau_i \in \mathcal{T}} (T_i - D_i)U_i + (U_{tot} - U_k)D_k}{m - U_{tot}} \tag{8.7}$$

Proof. This is an extension of the Theorem 3 in [Bar07]. The proof is similar to the proof of A_k^α, in which we use an over-approximation of the DBF function (as in Equation (6.1)) instead of the linearisation of the workload function. □

As a result, the smaller one between A_k^α and A_k^β is a safe upper bound on busy period length. Moreover, starting from $A_k = 0$, only those A_k values at which $\mathsf{DBF}_i(A_k + D_k)$ changes for some τ_i need to be considered. Finally, the RTA-LC-EDF test has a pseudo-polynomial time complexity: $O(n^3 \mathcal{L}_{max} D_{max} \mathcal{N})$[1] such that $\mathcal{L}_{max}$ is the maximum length of the extended problem window and $\mathcal{N}$ denotes the maximum number of A_k points checked.

In the case of $m = 1$, there is no pessimism in the computation of workload and interference for RTA-LC-EDF. Given that it takes only valid busy periods into account, RTA-LC-EDF is compatible with Spuri's RTA [Spu96] for EDF scheduling in the single processor and returns exact worst-case response time when $m = 1$.

Discussion To better understand the differences among the three tests Bar, BC and RTA-LC-EDF, it is useful to see what happens when they are applied to the case of $m = 1$.

As a matter of fact, BC provides only an upper bound to the response time of a task even for $m = 1$, because it analyses just the problem window and not the whole busy period. To the best of our knowledge, RTA-LC-EDF is the first response-time analysis for G-EDF that reduces to the exact Spuri's algorithm [Spu96] for single processors when $m = 1$.

The Bar schedulability test is based on the demand bound function, so it does not provide a response time. In fact, for the case of $m = 1$, Bar is equivalent to the classic single processor demand-bound analysis [BMR90], which is a necessary and sufficient test. Therefore, we can say that the main difference between Bar and RTA-LC-EDF is that the first provides a yes/no answer for schedulability, whereas the second additionally provides an upper-bound to the response time of a task (which becomes exact for $m = 1$).

On the other hand, we rely on more precise estimation of workload and interference to preserve the dominance over Bar and BC. The fine-grained workload computation of Algorithm 3 and 4 is more precise than the one used in Bar in the extended problem window. The total interference upper bound Ω_2 in Equation (8.2) is also a key factor and guarantees that RTA-LC-EDF's estimated interference on the target job is never more than the one computed by BC.

Furthermore, in Bar and RTA-LC-EDF, we need to advance the beginning of the problem window and this can result in additional pessimism: *such pessimism is inherent to limited CI techniques* and cannot be completely avoided; however, the computation of Ω_2 reduces it considerably.

8.1.4 RTA-LC-EDF-B

Here, we propose an over-approximation of the RTA-LC-EDF test of Theorem 22. We aim to improve the run-time efficiency, while preserving certain guarantees of the schedulability result.

Given the sub problem window $[s_0, s_1)$ (Figure 8.1), we further concentrate on its later part after the target job's release, that is $[a, s_1)$. We denote $y = s_1 - a$.

[1] The complexity formulated in the original paper [SL15] is more pessimistic.

Then, we define $\Omega(y) = \max_{\forall A_k}\{\Omega(A_k + y, A_k) - mA_k\}$. $\Omega(y)$ represents an upper bound on the total interference over interval $[a, s_1)$. In the following, we are interested in knowing if $\Omega(y)$ is large enough such that τ_k's target job cannot complete its worst-case execution in $[a, s_1)$; and we can still upper bound A_k values to the smaller one between A_k^α and A_k^β. In the new test, we skip the validity check of A_k (Equation (8.5)) for efficiency concern.

Moreover, we employ another bound on A_k that is dependent on the value of y. That is, given any y, we are going to find (if there exists) the first A_k such that the resulting interference does not leave enough space in the interval $[a, s_1)$ for the target job's worst-case execution.

Lemma 11. *In order to decide whether the target job is eligible to finish its worst-case execution within a time interval $[a, s_1)$ with length y, it is enough to consider A_k values $\leq A_k^y$ such that $\left\lfloor \frac{\Omega(A_k^y + y, A_k^y) - mA_k^y}{m} \right\rfloor > y - C_k$.*

Proof. If inequality $\left\lfloor \frac{\Omega(A_k^y + y, A_k^y) - mA_k^y}{m} \right\rfloor > y - C_k$ holds, it means that there exists $A_k = A_k^y$ such that the target job cannot finish execution inside $[a, a + y)$. □

Theorem 26 (RTA-LC-EDF-B)**.** *An upper bound of τ_k's WCRT is the solution of the following iterative procedure starting from $Y = \phi$ with $\phi = C_k$.*

$$Y \leftarrow \phi + \left\lfloor \frac{\Omega(Y)}{m} \right\rfloor \tag{8.8}$$

Proof. This can be proved using the same technique as in the proof of Theorem 22.

□

The complexity of RTA-LC-EDF-B is $O(n^3 D_{max}^2 \mathcal{N})$, where D_{max} is the maximum deadline among all tasks. However, the value of A_k^y is usually very small. This means that the overall number of A_k values checked in each step of Equation (8.8) can be very low too, and this would further benefit the run-time performance of RTA-LC-EDF-B.

As for the precision guarantee in RTA-LC-EDF-B, the following theorem proves that it still dominates Bar and BC.

Theorem 27. *RTA-LC-EDF-B $\succeq$ Bar and RTA-LC-EDF-B $\succeq$ BC.*

Proof. For any A_k and $C_k \leq y \leq D_k$, there is $\Omega(A_k + y, A_k) - mA_k \leq \Omega - mA_k$ and $\Omega(y) \leq \sum_{i \neq k} I_{i,k}(y)$, where Ω and $\sum_{i \neq k} I_{i,k}(y)$ are the total interference upper bounds by Bar and BC respectively. Then, following the same reasoning procedure as in the proof of RTA-LC-EDF's dominance over Bar and BC (Theorem 23), it holds that RTA-LC-EDF-B $\succeq$ Bar and RTA-LC-EDF-B $\succeq$ BC. □

In the end, since Bar is optimal in single processors, so is RTA-LC-EDF-B. That is, in case $m = 1$, a task is EDF schedulable if and only if the WCRT estimation returned by RTA-LC-EDF-B is no larger than that task's deadline.

Enhancing RTA-LC-EDF-B[2]

A more careful look into RTA-LC-EDF-B helps us further improve its efficiency and precision. At each step of the iteration as in Equation (8.8), the A_k^y also provides a lower bound on the A_k values to be considered for the next step: for any $A_k < A_k^y$, there exists $y' < y$ such that $\left\lfloor \frac{\Omega(A_k + y', A_k) - mA_k}{m} \right\rfloor + C_k \leq y'$. By taking this into account, the final complexity of RTA-LC-EDF-B will be $O(n^3(D_{max} + \mathcal{N})D_{max})$.

When $m = 1$, since there is no need to go through the refinement procesure, the overall complexity of RTA-LC-EDF-B becomes $O(n^2(D_{max} + \mathcal{N}))$. Although it does not necessarily return the exact WCRT,

[2] This enhancing version is not in the original paper [SL15].

RTA-LC-EDF-B is still an exact test when there is a single processor. To compute the exact WCRTs of a set of tasks in case of the single processor, the most efficient solution is by [GY14] and has the complexity $O(\mathcal{N}\mathcal{L}_{max} + n)$. Given the fact that very often $\mathcal{N} \gg n$ and $\mathcal{L}_{max} \gg D_{max}$, RTA-LC-EDF-B has a better run-time efficiency. Another advantage of RTA-LC-EDF-B is that it is able to estimate the WCRT of an arbitrary task τ_k with complexity $O(n(D_k + \mathcal{N}))$, however, the method in [GY14] has to always compute tasks' WCRTs all together.

8.2 Suspension-Aware Schedulability Analysis

8.2.1 Self-suspending tasks

Now, we are going to consider tasks that can suspend their executions. As in Section 2.1, by default we assume a task cannot suspend its execution voluntarily. Below we show how to extend a sporadic task model to take into account the self-suspension.

A self-suspending task τ_i is characterised by (C_i, D_i, T_i, S_i), where S_i denotes that a job of τ_i can at most suspend itself for S_i time units. Note that a job can suspend multiple times as long as the cumulative suspension time is no more than S_i, and it can begin or end with a suspension phase. We require that $C_i + S_i \leq D_i$.

A task that never suspends itself is also called a *computational* task for explicitly distinguishing it from self-suspending tasks. A computational task can be regarded as a special self-suspending task with $S_i = 0$. When there are both self-suspending and computational tasks in $\mathcal{T}$, we use $\mathcal{T}^s$ to denote the subset of self-suspending tasks and $\mathcal{T}^c$ to denote the subset of computational tasks. That is, $\mathcal{T} = \mathcal{T}^s \cup \mathcal{T}^c$ and $\mathcal{T}^s \cap \mathcal{T}^c = \emptyset$. For simplicity, we require that the size of $\mathcal{T}^c$ is at least $(m-1)$.

8.2.2 Suspension-aware schedulability: prior results

The state-of-the-art suspension-aware test for multiprocessor G-EDF scheduling in hard real-time systems is by Liu and Anderson [LA13], and we denote this test as LA. LA makes an extension from Bar and still relies on its extended problem window (see Figure 8.1).

A target job from τ_k is chosen for analysis and we use $s_{k,e} \in [0, S_k]$ to denote the cumulative suspension time of this job; such suspension time is needed to bound interference suffered by the target job within the extended problem window. For a computational task, $s_{k,e} = 0$ can be always assumed.

$$I_{i,k}^{NC} = \begin{cases} \left[\!\left[I_i^{NC}(A_k + D_k) \right]\!\right]^{A_k + D_k - C_k - s_{k,e} + 1} & \text{if } i \neq k \\ \left[\!\left[I_i^{NC}(A_k + D_k) - C_k \right]\!\right]^{[\![A_k + D_k - T_k]\!]_0} & \text{if } i = k \end{cases}$$

$$I_{i,k}^{CI} = \begin{cases} \left[\!\left[I_i^{CI}(A_k + D_k) \right]\!\right]^{A_k + D_k - C_k - s_{k,e} + 1} & \text{if } i \neq k \\ \left[\!\left[I_i^{CI}(A_k + D_k) - C_k \right]\!\right]^{[\![A_k + D_k - T_k]\!]_0} & \text{if } i = k \end{cases}$$

When bounding the total interference, a key difference between task sets with and without self-suspending tasks is that all self-suspending tasks can bring carry-in workload in the beginning of the extended problem window. In the end, together with the fact that there are at most $(m-1)$ computational tasks with CI workload, the total interference in the extended problem window becomes $\Omega =$

$$\sum_{\tau_i \in \mathcal{T}^s} I_{i,k}^{CI} + \sum_{\tau_j \in \mathcal{T}^c} I_{j,k}^{NC} + \sum_{\text{the } m-1 \text{ largest}} I_{j,k}^{DIFF}$$

Theorem 28 (Adapted from Theorem 2 in [LA13]). *A task set $\mathcal{T} = \mathcal{T}^s \cup \mathcal{T}^c$ is schedulable with G-EDF if, $\forall \tau_k \in \mathcal{T}$, $\forall s_{k,e} \in [0, S_k]$ and for any $0 \leq A_k < \overline{A}_k$, the following holds:*

$$\Omega \leq m(A_k + D_k - C_k - s_{k,e}) \tag{8.9}$$

with $\overline{A}_k = \frac{m \cdot (C_k + s_{k,e}) + \sum_{\tau_i \in \mathcal{T}} C_i}{m - U_{tot}} - D_k$.

8.2.3 RTA-LC-EDF(-B) with suspension-awareness

Here, we discuss how to apply RTA-LC-EDF(-B) to systems with self-suspensions. Again, a target job of τ_k with suspension time $s_{k,e}$ is considered.

From suspension-oblivious context to suspension-aware analysis, a series of adaptations are listed below.

- The suspension of a task does not contribute to its worst-case workload. Thus, the workload formulation of a task, which can be from either $\mathcal{T}^s$ or $\mathcal{T}^c$, in Algorithm 3 and Algorithm 4 is still valid.

- In order for the target job (with suspension) to successfully terminate, there should be enough processing time left for both its execution (C_k) and suspension ($s_{k,e}$). That is, when bounding the interference from workload, we should consider the target job's suspension.

$$I_{i,k}^{NC}(x, A_k) = \left[\!\left[W_i^{NC}(x, \mathcal{L}) \right]\!\right]^{x - C_k - s_{k,e} + 1}$$
$$I_{i,k}^{CI}(x, A_k) = \left[\!\left[W_i^{CI}(x, \mathcal{L}) \right]\!\right]^{x - C_k - s_{k,e} + 1}$$

- All self-suspending tasks are regarded as CI tasks. The total interference in the sub problem window is bounded by two estimates: Ω_1 in Equation (8.1) and Ω_2 in Equation (8.2), where Ω_1 is obtained by limiting the number of CI tasks in the beginning of sub problem window. As all self-suspending tasks are CI tasks now, we need to refine Ω_1's formulation.

$$\Omega_1 = \sum_{\tau_i \in \mathcal{T}^s} I_{i,k}^{CI}(x, A_k) + \sum_{\tau_j \in \mathcal{T}^c} I_{j,k}^{NC}(x, A_k) + \max_{\text{the } m-1 \text{ largest}} I_{j,k}^{DIFF}(x, A_k)$$

  Similarly, the total workload $W(x)$ within the sub problem window formulated in Equation (8.6) is refined as follows.

$$W(x) = \sum_{\tau_i \in \mathcal{T}^s} W_i^{CI}(x, \mathcal{L}) + \sum_{\tau_j \in \mathcal{T}^c} W_j^{NC}(x, \mathcal{L}) + \max_{\text{the } m-1 \text{ largest}} W_j^{DIFF}(x, \mathcal{L})$$

- In the suspension-aware context, all self-suspending tasks may bring CI workload into a problem window. Hence, we re-compute A_k^{α} as

$$\frac{C'_{\Sigma} + \sum_{\tau_i \in \mathcal{T}^s} C_i + \sum_{\tau_i \in \mathcal{T}} (T_i - C_i) U_i}{m - U_{tot}}$$

  and A_k^{β} as

$$\frac{C'_{\Sigma} + \sum_{\tau_i \in \mathcal{T}^s} C_i + \sum_{\tau_i \in \mathcal{T}} (T_i - D_i) U_i + (U_{tot} - U_k) D_k}{m - U_{tot}}$$

with C'_Σ denoting sum of the $(m-1)$ largest WCETs among computational tasks.

On the other hand, to decide if the interference $\Omega(y)$ is too large so that the target job's execution, together with its self-suspension, exceeds the interval $[a, s_1)$, it is enough to bound A_k by the following A_k^y with

$$\left\lfloor \frac{\Omega(A_k^y + y, A_k^y) - mA_k^y}{m} \right\rfloor > y - C_k - s_{k,e}.$$

- The last step to adapt RTA-LC-EDF and RTA-LC-EDF-B for **S**uspension-**A**wareness (SA) context is a new starting point, which should take into account the suspension time $s_{k,e}$, for their respective RTA procedures. That is, $\phi = C_k + s_{k,e}$ for Equation (8.4) and Equation (8.8).

In the end, the adapted tests are denoted as RTA-LC-EDF-SA and RTA-LC-EDF-SA-B. Instead of enumerating all possible $s_{k,e}$ in $[0, S_k]$ for the WCRT of the target task, we prove that it is enough to only consider the job with maximum suspension time S_k.

Theorem 29. *The WCRT of τ_k can be upper bounded when its target job has $s_{k,e} = S_k$.*

Proof. Let us first have a look at RTA-LC-EDF-SA. Suppose that $\phi_1 = C_k + s_{k,e}$ and $\phi_2 = C_k + s'_{k,e}$ are two starting points (let us say $A_k = 0$) for Equation (8.4) in the suspension-aware context and $s_{k,e} > s'_{k,e}$. Given two sub problem windows with length X_1 and X_2 corresponding to ϕ_1 and ϕ_2 respectively and $X_1 - X_2 = s_{k,e} - s'_{k,e}$, for any task, the upper bound on its interference in the two cases is $X_1 - \phi_1 + 1 = X_2 - \phi_2 + 1$. That means, if we choose an arbitrary iteration step, the difference between the results starting from ϕ_1 and ϕ_2 is at least $(s_{k,e} - s'_{k,e})$. By assuming $s_{k,e} = S_k$, the resulting WCRT estimation would be a safe upper bound.

For RTA-LC-EDF-SA-B, a similar proof can be conducted. □

In the end, we prove the dominance of the new suspension-aware algorithms over LA.

Theorem 30. *RTA-LC-EDF-SA $\succeq$ RTA-LC-EDF-SA-B $\succeq$ LA.*

Proof. Here we only sketch the proof for RTA-LC-EDF-SA-B $\succeq$ LA. Similarly to the suspension-oblivious situation, when explicitly taking the self-suspending time $s_{k,e}$ into account, we have $\Omega(A_k + y, A_k) - mA_k \leq \Omega - mA_k$; $\Omega(A_k + y, A_k)$ and Ω are estimated by RTA-LC-EDF-SA-B and LA (Theorem 28) respectively. Thanks to the more precise estimation of workload and interference, RTA-LC-EDF-SA-B $\succeq$ LA. □

8.3 Evaluation

In this section we evaluate the performance of different schedulability tests with or without explicitly taking task suspension time into account. For simplicity, in this section, we use A and B as abbreviations for RTA-LC-EDF and RTA-LC-EDF-B, respectively; and SA-A and SA-B denote the suspension-aware counterparts. Recently, [Lee14] developed a "divide and conquer"technique and applied it to BC, denoted as DC-BC (more specifically, we refer to the NEW-$\mathsf{C}_{\mathsf{EDF}}$ in [Lee14]). In brief, DC-BC judges the target task τ_k to be schedulable if there exists an integer pair configuration $(c', l) \in [0, C_k] \times [0, D_k]$ dividing τ_k into two parts such that both parts satisfy certain interference-based conditions derived from BC. We include DC-BC in the comparison.

8.3.1 Tasks without self-suspension

Each task set in the simulation is characterised by a tuple (m, n, U_{tot}) such that m is the number of processors, n is the number of tasks and U_{tot} is the total task utilisation. We consider $m \in \{2, 4, 8\}$, and

we set $n \in \{2 \cdot m, 4 \cdot m, 8 \cdot m, 10 \cdot m\}$. For each (m, n, U_{tot}) configuration, we randomly generate 1000 task sets. For a task set with some U_{tot}, tasks' utilisations are generated according to Randfixedsum algorithm [ESD10]. Task periods are randomly sampled with uniform distribution in the range $[10, 1000]$. Then, the WCET of a task τ_i is simply computed as $T_i \cdot U_i$ and the relative deadline D_i is randomly chosen with uniform distribution in a range $[0.8 \cdot T_i, T_i]$.

For each generated task set, different schedulability tests are applied, and we record the the number of schedulable task sets returned by each test. Results are reported in Figure 8.4, 8.5 and 8.6. The x-axis denotes the total task utilisation U_{tot} and the y-axis represents the percentage of schedulable task sets detected by different tests. The line marked with Bar/BC counts the task sets that are found schedulable by at least one of the two.

Clearly RTA-LC-EDF and RTA-LC-EDF-B substantially improve over other tests. DC-BC may slightly perform better than RTA-LC-EDF when the task set size is small (i.e. $\frac{n}{m} = 2$). This leaves us an motivation for the investigation of applying "divide and conquer" idea to RTA-LC-EDF in the future.

8.3.2 Tasks with self-suspension

We evaluated the performance of our new suspension-aware tests SA-A and SA-B. Task sets are generated in the same way as before; given any task τ_i, its maximum suspension time S_i is then chosen in the range $[0.5 \cdot C_i, C_i]$. Besides comparing with the state-of-the-art suspension-aware test LA, we also apply Bar, BC, RTA-LC-EDF(-B) and DC-BC in the suspension-oblivious way, i.e., by simply treating self-suspending time as task execution.

Results are in Figure 8.7, 8.8 and 8.9. Notice that LA is very pessimistic and gives positive results for very few task sets. The suspension-aware version of RTA-LC-EDF further improves over the suspension-oblivious version.

8.3.3 Run-time efficiency

As we have seen from simulation results, the performances of RTA-LC-EDF and of its over-approximation RTA-LC-EDF-B are rather close to each other. However, RTA-LC-EDF-B has a better run-time performance, as shown in Figure 8.3, where we report the running time of the two algorithms on task sets with $m = 8$, $n = 10 \cdot m$ and different utilisations. Notice that the running time of RTA-LC-EDF-B is up to 100 times shorter than that of RTA-LC-EDF.

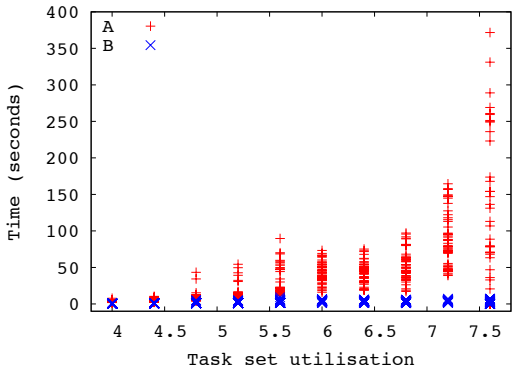
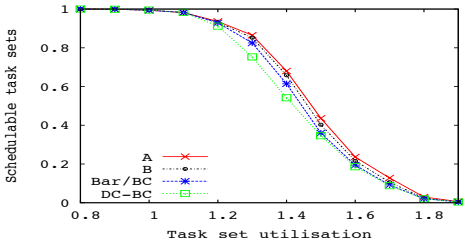
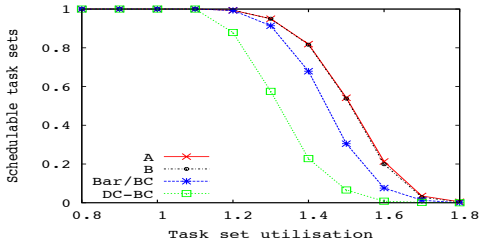


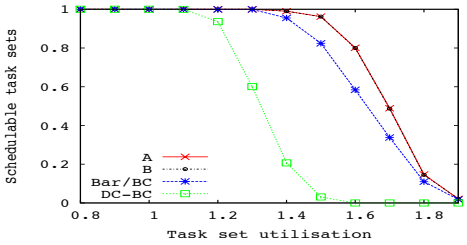
Figure 8.3: Run-time comparison between new tests



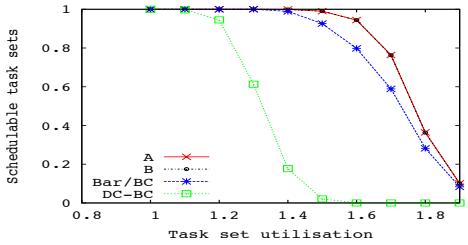
(a) $\frac{n}{m} = 2$



(b) $\frac{n}{m} = 4$

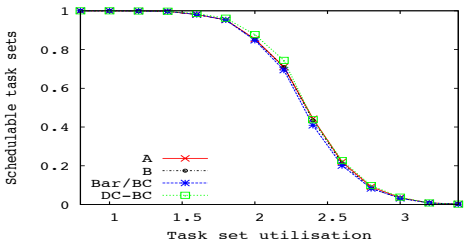


(c) $\frac{n}{m} = 8$

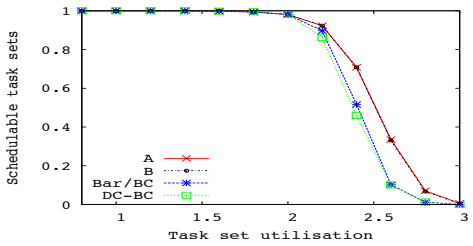


(d) $\frac{n}{m} = 10$

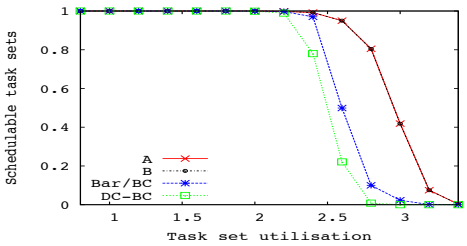
Figure 8.4: Tests on 2 processors (**without** self-suspension)



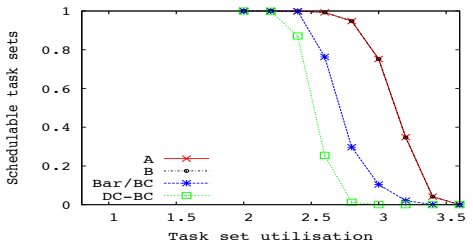
(a) $\frac{n}{m} = 2$



(b) $\frac{n}{m} = 4$



(c) $\frac{n}{m} = 8$

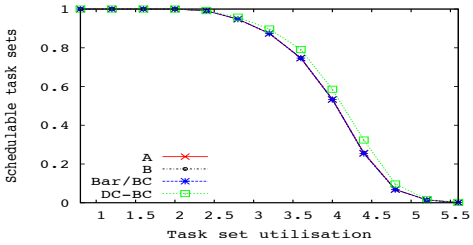


(d) $\frac{n}{m} = 10$

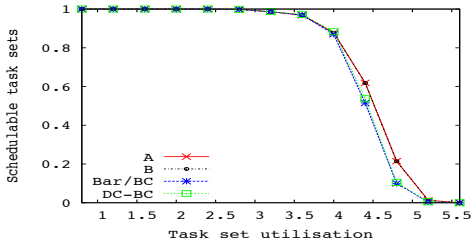
Figure 8.5: Tests on 4 processors (**without** self-suspension)

8.4 Conclusion

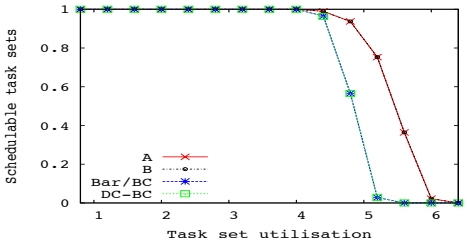
In this chapter, we provided more accurate algorithms for computing the response times of real-time tasks scheduled by G-EDF. We also extended the methodology to self-suspending tasks. Our algorithms strictly dominate the state-of-the-art algorithms, and substantial performance improvements are confirmed by



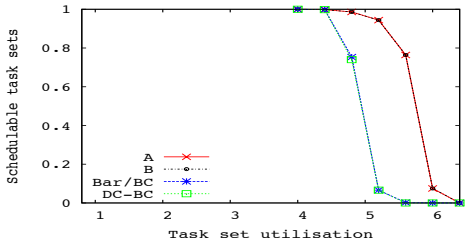
(a) $\frac{n}{m} = 2$



(b) $\frac{n}{m} = 4$

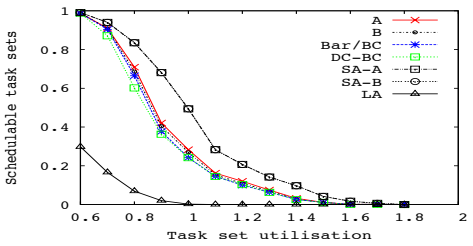


(c) $\frac{n}{m} = 8$

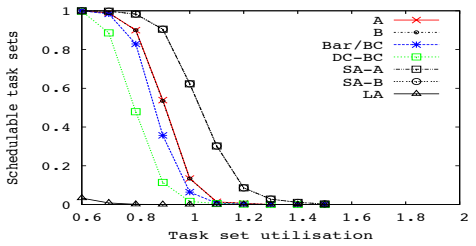


(d) $\frac{n}{m} = 10$

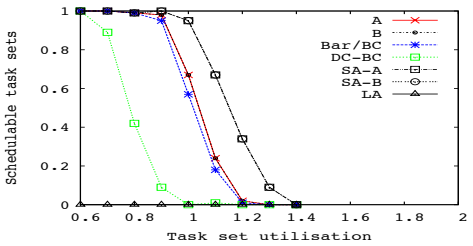
Figure 8.6: Tests on 8 processors (**without** self-suspension)



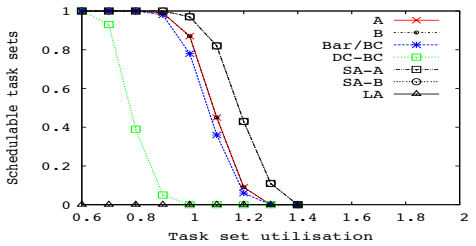
(a) $\frac{n}{m} = 2$



(b) $\frac{n}{m} = 4$



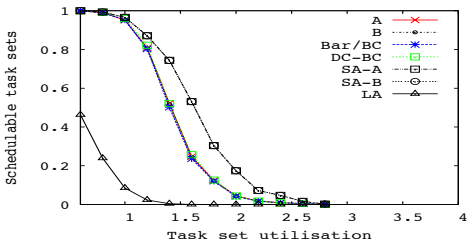
(c) $\frac{n}{m} = 8$



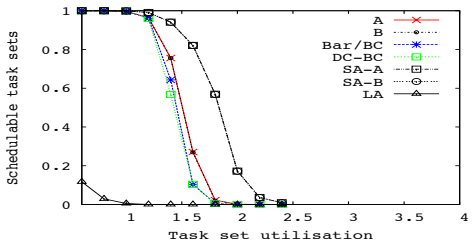
(d) $\frac{n}{m} = 10$

Figure 8.7: Tests on 2 processors (**with** self-suspension)

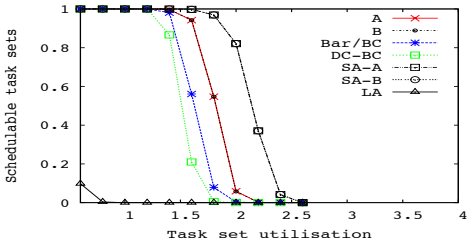
simulation results. In the future, we plan to extend the algorithm to other task models, taking into account interaction through shared resources.



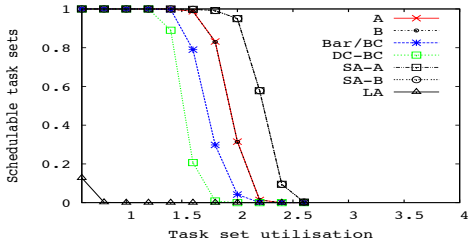
(a) $\frac{n}{m} = 2$



(b) $\frac{n}{m} = 4$

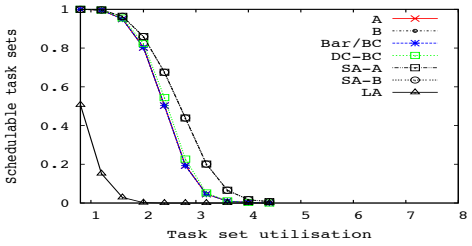


(c) $\frac{n}{m} = 8$

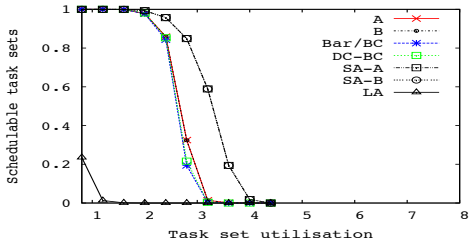


(d) $\frac{n}{m} = 10$

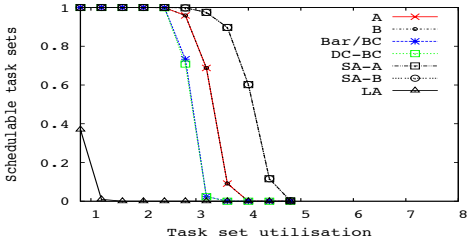
Figure 8.8: Tests on 4 processors (**with** self-suspension)



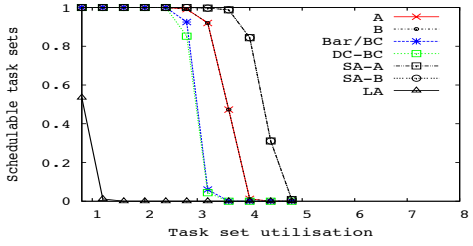
(a) $\frac{n}{m} = 2$



(b) $\frac{n}{m} = 4$



(c) $\frac{n}{m} = 8$



(d) $\frac{n}{m} = 10$

Figure 8.9: Tests on 8 processors (**with** self-suspension)

Chapter 9

Task Parameter Scalability Problem

In Chapter 5, the proposed exact G-FP schedulability analysis is built upon Linear Hybrid Automata (LHA), and relies on the continuous time schedule; that is, the $r_{i,j}$, $f_{i,j}$ and $d_{i,j}$ for a job $J_{i,j}$ can be any non-negative values. Whereas from Chapter 6 to 8, when talking about the analytical schedulability analysis of multiprocessor systems, the discrete time schedule is considered, that is, the $r_{i,j}$, $f_{i,j}$ and $d_{i,j}$ for a job $J_{i,j}$ must be non-negative integers.

The discrete time assumption reflects the fact that processing time in a computing platform can always be divided into discrete *ticks*. Such an assumption is very common in real-time systems. For instance, The Response Time Analysis (RTA) [ABR+93] is a foundmental methodology for real-time schedulability analysis and it relies on the assumption that time is divided into integers.

A good feature for schedulability analysis in continuous time domain is that, given a schedulable task set, by scaling up all tasks' parameters, the resulting task set is still schedulable. But is this also *true* when a discrete time schedule is considered? We believe such a property matters, and it concerns with the robustness of the scheduling policy and the schedulability analysis.

In this chapter, we propose the concept of task parameter scalability (p-scalability) in a real-time system by investigating the facts (in discrete time domain) that for some scheduling algorithms the schedulability of a task set is not preserved when its task parameters are scaled. Examples are the multiprocessor Global Fixed Priority (G-FP) scheduling of periodic or sporadic tasks and the uniprocessor Fixed Priority scheduling of real-time tasks with self-suspension.

For such scheduling algorithms, we discuss principles for the safe schedulability analysis. In particular, we demonstrate that state-of-the-art schedulability tests for G-FP scheduling are not safe and a new methodology is proposed for the G-FP schedulability analysis by taking the p-scalability property into account.

When talking about the p-scalability property, by default the discrete time domain is assumed. Without loss of generality, we will focus on Fixed Priority (FP) systems.

The content in this chapter is based on [SLb], and all results hold for both sporadic and periodic tasks.

9.1 Task Parameter Scalability

In this section, we are going to formally present the definition of task parameter scalability.

Given a task $\tau_i = (C_i, D_i, T_i)$ and a positive integer α, the *α-scaling* of τ_i is a task characterised by $(\alpha \cdot C_i, \alpha \cdot D_i, \alpha \cdot T_i)$ and is denoted as $\alpha \cdot \tau_i$. As an example, for a task $\tau_i = (5, 10, 10)$, its 10-scaling is $10 \cdot \tau_i = (50, 100, 100)$. We use the term *any-scaling* to denote an arbitrary α-scaling task of τ_i.

For a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, its α-scaling is $\alpha \cdot \mathcal{T} = \{\alpha \cdot \tau_1, \ldots, \alpha \cdot \tau_n\}$. An any-scaling task set can be similarly defined as the any-scaling task.

Given a task set $\mathcal{T}$ and a scheduling policy $\mathfrak{S}$ such that $\mathcal{T}$ is schedulable by $\mathfrak{S}$, does the schedulability result hold for $\alpha \cdot \mathcal{T}$ ($\forall \alpha \in \mathbb{Z}^+$)?

The answer of the above question does not come for free and this motivates us to explore the concept of *task parameter scalability*.

Definition 15 (Task parameter scalability)**.** *We say a scheduling algorithm $\mathfrak{S}$ is task parameter scalable (p-scalable) if for any task set $\mathcal{T}$ that is schedulable under $\mathfrak{S}$, its any-scaling task set $\alpha \cdot \mathcal{T}$ ($\forall \alpha \in \mathbb{Z}^+$) is also schedulable.*

For a p-scalable system, it is enough to analyse the schedulability of the set of tasks with parameters scaled down from original values. On the other hand, for a non p-scalable system, schedulability analysis can be tricky, for which we are going to discuss later. At first, we would like to investigate the p-scalability property of several well-known FP systems.

9.1.1 Uniprocessor FP scheduling

The uniprocessor FP scheduled system is the most well-studied topic in the literature. A sufficient and necessary test is in [ABR+93] and it computes the exact WCRT of a task, which can be either periodic or sporadic. Given the periodic or sporadic task τ_i, its exact WCRT can be obtained through the following iterative procedure starting with $X = C_i$.

$$X = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{X}{T_j} \right\rceil C_j$$

Theorem 31. *Uniprocessor FP scheduling is p-scalable.*

Proof. For any task $\tau_i \in \mathcal{T}$ with WCRT R_i, we use $R_{\alpha \cdot i}$ to denote the WCRT of its α-scaling $\alpha \cdot \tau_i \in \alpha \cdot \mathcal{T}$.

Given the task $\tau_i \in \mathcal{T}$ and X, and $\alpha \cdot \tau_i \in \alpha \cdot \mathcal{T}$ and $\alpha \cdot X$, the following relation holds:

$$\alpha \cdot \mathrm{C_i} + \sum_{j=1}^{i-1} \left\lceil \frac{\alpha \cdot \mathrm{X}}{\alpha \cdot \mathrm{T_j}} \right\rceil \alpha \cdot \mathrm{C_j} = \alpha \cdot (\mathrm{C_i} + \sum_{\mathrm{j=1}}^{\mathrm{i-1}} \left\lceil \frac{\mathrm{X}}{\mathrm{T_j}} \right\rceil \mathrm{C_j}).$$

This means that for any $\alpha \cdot \tau_i \in \alpha \cdot \mathcal{T}$, there is $R_{\alpha \cdot i} = \alpha \cdot \mathrm{R_i}$. That is, if τ_i is schedulable ($R_i \leq D_i$), then $\alpha \cdot \tau_i$ is also schedulable ($\alpha \cdot \mathrm{R_i} \leq \alpha \cdot \mathrm{D_i}$). In conclusion, uniprocessor FP scheduling is p-scalable. □

9.1.2 Multiprocessor G-FP scheduling

Theorem 32. *Multiprocessor G-FP scheduling is not p-scalable.*

Proof. We use a counter-example to demonstrate that G-FP is not p-scalable. Let us assume a task set $\mathcal{T}$ of 4 sporadic tasks running on 2 processors: $\tau_1 = (1, 4, 4)$, $\tau_2 = (1, 3, 3)$, $\tau_3 = (1, 3, 3)$ and $\tau_4 = (1, 2, 2)$. The schedulability of this task set can be confirmed by applying the sufficient schedulability test like RTA-CE (Theorem 21).

Given a schedulable task set with sporadic tasks, if tasks become periodic, they are still schedulable.

However, $10 \cdot \mathcal{T}$ is not schedulable. For example, when τ_2, τ_3 and τ_4 are released together at time point 0, whereas τ_1 is released at time point 1, such a release pattern will result in τ_4's deadline miss and this has nothing to do with sporadic or periodic task activations.

Hence, the G-FP scheduling is not p-scalable.

□

9.1.3 Uniprocessor FP scheduling of self-suspending tasks

As we have seen in Section 8.2.1, in some systems, a task may be allowed to self-suspend its execution, and it is called a *self-suspending* task. Different from the case of non self-suspending tasks as in Section 9.1.1, the exact schedulability condition of uniprocessor FP scheduling of real-time tasks with self-suspension is unknown.

In particular, we adopt a simple model such that a self-suspending task is depicted as $(C_{i,1}, C'_{i,1}, S_{i,1}, S'_{i,1}, C_{i,2}, C'_{i,2}, D_i, T_i)$, where there are two execution sessions that are separated by a suspension session and the terms D_i and T_i are the same as in non self-suspending tasks. Each execution session is specified by its lower bound ($C_{i,1}$ and $C_{i,2}$) and upper bound ($C'_{i,1}$ and $C'_{i,2}$) on the execution time; $S_{i,1}$ and $S'_{i,1}$ are the lower and upper bounds on a task's self-suspending time. Although here only a suspension session is allowed, multiple suspensions of a task can be extended easily. A non self-suspending task can be regarded as a special self-suspending tasks with $S_{i,1} = S'_{i,1} = 0$.

Theorem 33. *The uniprocessor FP scheduling of a set of self-suspending tasks is not p-scalable.*

Proof. This can be proved by a counter-example. Let us see a task set $\mathcal{T}$ with two tasks: $\tau_1 = (C_{1,1} = 1, C'_{1,1} = 1, S_{1,1} = 1, S'_{1,1} = 2, C_{1,2} = 1, C'_{1,2} = 1, D_1 = 8, T_1 = 8)$ and $\tau_2 = (C_{2,1} = 1, C'_{2,1} = 1, S_{2,1} = 2, S'_{2,1} = 2, C_{2,2} = 1, C'_{2,2} = 1, D_2 = 5, T_2 = 5)$. τ_1 and τ_2 are running upon a uniprocessor platform, and τ_1 has a higher priority.

In this simple system, an enumeration of all patterns of task arrivals and execution/self-suspension interleavings is possible, and the task τ_2 is schedulable. As an example, we only present here a particular case such that τ_1 and τ_2 arrives simultaneously (let us say at time 0).

- For the time unit $[0, 1)$, τ_1 occupies the processor.
- For $[1, 2)$ τ_1 goes to suspension and τ_2 executes.
- For $[2, 3)$, there are two cases: 1) τ_1 starts its 2nd execution session, and 2) τ_1 still self-suspends. In both cases, τ_2 goes into self-suspension and τ_1 will not interfere τ_2's 2nd execution phase. In the end, τ_2 will meet its deadline.

However, when it comes to $\alpha \cdot \mathcal{T}$, possible interleavings between $\alpha \cdot \tau_1 = (10, 10, 10, 20, 10, 10, 80, 80)$ and $\alpha \cdot \tau_2 = (10, 10, 20, 20, 10, 10, 50, 50)$ can result in the deadline miss of $\alpha \cdot \tau_2$. Suppose that $\alpha \cdot \tau_2$ is released at time 0 and $\alpha \cdot \tau_1$ is released one time unit later, and both tasks self-suspend by their respective upper bounds, finally $\alpha \cdot \tau_2$ will miss its deadline. The corresponding schedule is depicted as in Figure 9.1. Thus, the FP scheduling of self-suspending tasks is not p-scalable.

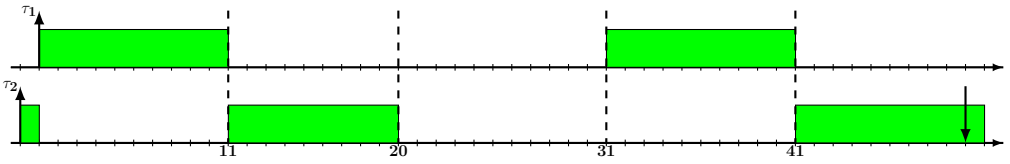


Figure 9.1: The scheduling of self-suspending tasks is not p-scalable

□

9.2 The Schedulability Analysis for non P-Scalable Scheduling

9.2.1 Certain and uncertain tests

Given any scheduling policy, the core is to check the schedulability of a task set under it. The concept of p-scalability brings new considerations for designing schedulability tests.

At first, for a non p-scalable scheduling policy, *the exact test seems to be less attractive*. Given an exact test $\mathfrak{T}$ and suppose that $\mathfrak{T}$ verifies a task set $\mathcal{T}$ to be schedulable, this does not bring the guarantee for the schedulability of the any-scaling of $\mathcal{T}$.

As a consequence, we differentiate schedulability tests for a non p-scalable system into two classes:

- A test $\mathfrak{T}$ is said to be *certain* if for any task set $\mathcal{T}$ that is decided to be schedulable by $\mathfrak{T}$, its any-scaling $\alpha \cdot \mathcal{T}$ $(\forall \alpha \in \mathbb{Z}^+)$ is also schedulable by $\mathfrak{T}$;
- otherwise, the test is *uncertain*.

In the end, for a non p-scalable scheduling algorithm, a certain test is safe to use. When checking the schedulability of task sets under a non p-scalable algorithm, since the exact test is uncertain, a less pessimistic certain test will be the best that can be expected.

In the following, we will take the multiprocessor G-FP scheduling as an example, and demonstrate how to perform certain schedulability analysis in a non p-scalable system.

9.2.2 The schedulability analysis for G-FP: prior results

In this part, we briefly recall the state-of-the-art tests RTA-LC, RTA-CE and DA-LC for schedulability check of task sets under G-FP policy, which is not p-scalable. More details on the analytical techniques for G-FP schedulability analysis can be found in Chapter 6 and Chapter 7. For simplicity, we assume tasks have constrained deadlines.

A *target* task τ_k is chosen for the schedulability check; more specifically, we can focus on one arbitrary job of τ_k that is called the *target* job. When analysing the schedulability of such a job, a problem window is assumed. The problem window is a time interval $[a, b)$ such that a coincides with the release time of the target job and b is the corresponding absolute deadline. A sub problem window of is a time interval $[a, s)$ with $s \leq b$, and we denote $x = s - a$.

Given a sub problem window, the *workload* of a higher priority task τ_i represents the maximum amount of execution can be conducted by τ_i over this time interval. In order to check the schedulability of the target task, we need to convert workload to the interference upon the target job. The *interference* from a higher priority task τ_i denotes the cumulative length of all time intervals such that the target job of τ_k is active but cannot execute, while τ_i is executing. The computation of the exact interference is very hard, we will focus on the upper bound of it. Since almost all state-of-the-art results for G-FP consider only the over-approximate interference, we would abuse the use of the term "interference" to denote also the upper bound of interference. According to the observation in [BCL05], it is enough to upper bound a higher priority task τ_i's interference $I_{i,k}(x)$ over a time interval x as follows.

$$0 \leq I_{i,k}(x) \leq (x - C_k + 1) \tag{9.1}$$

In order to more precisely estimate the interference caused upon the target job, interfering tasks are differentiated into two classes: *carry-in* (CI) tasks and *non-carry-in* (NC) tasks. Given the problem window $[a, b)$, a task is said to be a CI task if it has a job released before the beginning of the window and such a job is still active at time point a; otherwise, it is a NC task.

According to Theorem 20, to estimate the WCRT of τ_k, we only need to consider situations with up to $(m - 1)$ CI tasks.

For a NC task, the workload upper bound can be easily computed as follows.

$$W_i^{NC}(x) = \left\lfloor \frac{x}{T_i} \right\rfloor \cdot C_i + [\![x \bmod T_i]\!]^{C_i} \tag{9.2}$$

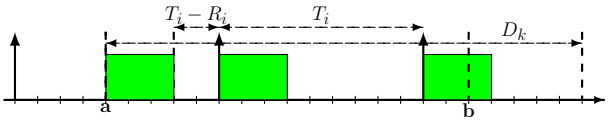


Figure 9.2: The worst-case arrival pattern for $W_i^{CI}(x)$

On the other hand, Figure 9.2 depicts the worst-case scenario for resulting the maximum workload of a CI task, and we adapt the formulation in Equation (7.1) to estimate the CI workload.

$$W_i^{CI}(x) = W_i^{NC}(\llbracket x - x_p \rrbracket_0) + \llbracket x \rrbracket^{C_i} \tag{9.3}$$

with $x_p = C_i + T_i - R_i$.

Then, we derive the interference from a higher priority task τ_i on the target one, according to the property in Equation (9.1) and workload formulation in Equation (9.2) and (9.3).

$$I_{i,k}^{NC}(x) = \llbracket W_i^{NC}(x) \rrbracket^{(x-C_k+1)} \tag{9.4}$$

$$I_{i,k}^{CI}(x) = \llbracket W_i^{CI}(x) \rrbracket^{(x-C_k+1)} \tag{9.5}$$

Given a time interval with length x, the total interference is defined.

$$\Omega_k(x) = \sum_{i \neq k} I_{i,k}^{NC}(x) + \sum_{\text{the } (m-1) \text{ largest}} I_{i,k}^{DIFF}(x) \tag{9.6}$$

where $I_{i,k}^{DIFF}(x) = I_{i,k}^{CI}(x) - I_{i,k}^{NC}(x)$.

Finally, the interference upon the target job within a sub problem window with length x can be upper bounded by $\left\lfloor \frac{\Omega_k(x)}{m} \right\rfloor$.

When it comes to the schedulability analysis, RTA-LC computes the response time of each task by incrementally enlarging the sub problem window and estimating the interference at each iterative step.

Lemma 12 (RTA-LC)**.** *For the target task $\tau_k \in \mathcal{T}$, its WCRT is bounded by the solution of following iteration, starting with $X = C_k$.*

$$X \leftarrow \left\lfloor \frac{\Omega_k(X)}{m} \right\rfloor + C_k \tag{9.7}$$

In Chapter 7, the test RTA-CE applies a Carry-in Enumeration (CE) technique to RTA-LC by explicitly enumerating the possible sets of CI tasks in prior to the RTA procedure.

Lemma 13 (RTA-CE)**.** *The upper bound of WCRT of task $\tau_k \in \mathcal{T}$ is*

$$R_k = \max_{\mathcal{T}^{CI} \in \mathcal{Z}} \left(R_k^{\mathcal{T}^{CI}} \right)$$

where $\mathcal{Z}$ is the set of all possible CI task sets and given a CI task set $\mathcal{T}^{CI}$, $R_k^{\mathcal{T}^{CI}}$ is computed by the following iteration

$$X = \left\lfloor \frac{\Omega_k(\mathcal{T}^{CI}, X)}{m} \right\rfloor + C_k$$

with

$$\Omega_k(\mathcal{T}^{CI}, x) = \sum_{\tau_i \notin \mathcal{T}^{CI}} I_{i,k}^{NC}(x) + \sum_{\tau_i \in \mathcal{T}^{CI}} I_{i,k}^{CI}(x).$$

Instead of an iterative procedure, DA-LC measures the total interference within a problem window directly.

Lemma 14 (DA-LC)**.** *For the target task $\tau_k \in \mathcal{T}$, τ_k is schedulable if*

$$D_k \geq \left\lfloor \frac{\Omega_k(D_k)}{m} \right\rfloor + C_k.$$

9.2.3 Certain schedulability analysis for G-FP

In this part, we first demonstrate that the G-FP schedulability tests discussed in prior are all not certain. Then, we analyse the causes of the uncertainty and provide a new methodology for certain schedulability analysis of G-FP scheduling.

Theorem 34. *The G-FP schedulability tests RTA-LC, RTA-CE and DA-LC are uncertain.*

Proof. The counter-example in the proof of Theorem 32 can be re-used here to prove the uncertainty of these tests. In fact, by applying them for randomly generated task sets and corresponding scalings, the theorem can be easily confirmed. □

Although it is enough to show a test's uncertainty through a counter-example, we would like to provide a brief analysis on factors causing the uncertainty. We start from RTA-LC.

Given a task $\tau_k \in \mathcal{T}$ and a time interval with length L, we build the scenario containing τ_k's α-scaling $\alpha \cdot \tau_k \in \alpha \cdot \mathcal{T}$ and the time interval with length $\alpha \cdot L$. Let us say, by RTA-LC, the interference upon τ_k subject to L is $\mathcal{I}$, and the interference upon $\alpha \cdot \tau_k$ subject to $\alpha \cdot L$ is $\mathcal{I}'$.

Given a mathematical function $\mathcal{F}$, we use the form $\mathcal{F}(L, \tau_i)$ to denote the computation result of $\mathcal{F}$ with respect to the input L and task τ_i. We say $\mathcal{F}$ is *certainty-friendly*, if given an input L, it always holds that $\forall \alpha \in \mathbb{Z}^+, \mathcal{F}(\alpha \cdot L, \alpha \cdot \tau_i) \leq \alpha \cdot \mathcal{F}(L, \tau_i)$; otherwise, $\mathcal{F}$ is said to be *certainty-unfriendly*.

If all mathematical functions appearing in RTA-LC, from Equation (9.1) to Equation (9.7), for estimating the workload and interference are certainty-friendly, then there would be $\mathcal{I}' \leq \alpha \cdot \mathcal{I}$, and RTA-LC will be a certain test. As this is not the fact, let us detect the certainty-unfriendly parts in RTA-LC.

- The interference upper bound $x - C_k + 1$ in Equation (9.1) is certainty-friendly, as there is

$$(\alpha \cdot L - \alpha \cdot C_k + 1) \leq \alpha \cdot (L - C_k + 1).$$

- The NC workload estimation in Equation (9.2) is certainty-friendly. In fact, it holds that $\alpha \cdot W_i^{NC}(L) = W_{\alpha \cdot i}^{NC}(\alpha \cdot L)$, where $\alpha \cdot i$ emphasises that the corresponding task is $\alpha \cdot \tau_i$.

  Whereas for the CI workload estimated in Equation (9.3), we need to consider different conditions. If there is $R_{\alpha \cdot i} \leq \alpha \cdot R_i$ (again, $\alpha \cdot i$ is used for the explicit denotation for task $\alpha \cdot \tau_i$), then $W_i^{CI}(L)$ is certainty-friendly; otherwise, $W_i^{CI}(L)$ may be certainty-unfriendly.

- Due to the certainty-friendliness of $L - C_k + 1$ and $W_i^{NC}(L)$, the NC interference estimated in Equation (9.4) is also certainty-friendly.

  However, the certainty-friendliness of the CI interference in Equation (9.5) depends on the relation between R_i and $R_{\alpha \cdot i}$, so does the total interference in Equation (9.6).

- Finally, it is the turn to the RTA-LC's iterative function in Equation (9.7). Let us first recall the following relation for the *floor* operator [flo]:

$$\forall y, \alpha, m \in \mathbb{Z}^+ \quad \left\lfloor \frac{\alpha \cdot y}{m} \right\rfloor \geq \alpha \cdot \left\lfloor \frac{y}{m} \right\rfloor.$$

  This says that the *floor* operator is not certainty-friendly, and this results in a certainty-unfriendly iterative procedure for RTA-LC to upper bound a task's response time as in Equation (9.7).

Now, we propose a new RTA procedure with a certain schedulability result.

Theorem 35 (Certain RTA-LC)**.** *For the target task $\tau_k \in \mathcal{T}$, its WCRT is bounded by the solution of following iteration, starting with $X = C_k$.*

$$X \leftarrow \left\lceil \frac{\Omega_k(X)}{m} \right\rceil + C_k \tag{9.8}$$

Proof. As the new iteration in Equation (9.8) will never converge into a fixed point that is smaller than Equation (9.7)'s, the new test indeed upper bounds τ_k's WCRT.

Implicitly, a task τ_k's WCRT upper bound is also denoted by R_k, and for $k \leq m$ there is $R_k = C_k$, as we only apply the RTA to tasks other than the m highest priority ones.

Then, certainty of the new test can be proved if given the task τ_k such that $\forall i < k \;\; R_{\alpha \cdot i} \leq \alpha \cdot R_i$, it holds that $R_{\alpha \cdot k} \leq \alpha \cdot R_k$.

Because that $R_{\alpha \cdot i} \leq \alpha \cdot R_i$, the CI interference $I_{i,k}^{CI}(L)$ in Equation (9.5) and the total interference $\Omega_k(L)$ in Equation (9.6) become certainty-friendly.

On the other hand, in the iterative procedure, we replace the *floor* operator with the *ceiling*, which is certainty-friendly [flo]:

$$\forall y, \alpha, m \in \mathbb{Z}^+ \;\; \left\lceil \frac{\alpha \cdot y}{m} \right\rceil \leq \alpha \cdot \left\lceil \frac{y}{m} \right\rceil .$$

As a result, there will be $R_{\alpha \cdot k} \leq \alpha \cdot R_k$, and the RTA test in Theorem 35 is certain.

□

Still, the CE technique in Chapter 7 can be applied to Certain RTA-LC and this will result in a Certain RTA-CE.

Similarly, a certain version of DA-LC is as follows.

Theorem 36 (Certain DA-LC)**.** *For the target task $\tau_k \in \mathcal{T}$, τ_k is schedulable if*

$$D_k \geq \left\lceil \frac{\Omega_k(D_k)}{m} \right\rceil + C_k.$$

9.3 Conclusion

In this chapter, we formulated the task parameter scalability (p-scalability) problem in a real-time system. Such a property is meaningful for the safe timing analysis. Unfortunately, many well-known systems are not p-scalable. That is, given a schedulable system, by scaling all task parameters, the resulting system may be not schedulable.

We then discussed how to solve challenges for schedulability analysis in a non p-scalable system. As it is shown, state-of-the-art tests for multiprocessor G-FP scheduling are not really safe. In the end, a new methodology is developed for G-FP schedulability analysis that takes the p-scalability into account.

Although this chapter deals with only fixed priority scheduling, similar issues also appear in dynamic priority systems. For example, the tests BC (Theorem 15) and RTA-LC-EDF (Theorem 22) for multiprocessor global Earliest Deadline First (EDF) scheduling obviously do not follow the safe principles regarding p-scalability. It will be interesting to further explore the p-scalability property in different real-time systems.

Chapter 10

Conclusion

Through this thesis, a variety of problems have been investigated to integrate the use of analytical approach and model-based approach for real-time schedulability analysis.

Classic theorems from the literature of real-time scheduling can be applied to the modeling of real-time systems so as to optimise the exploration of the state space. As in Chapter 3, a naive parametric analysis in the LHA model fails to return meaningful results and this is fixed by taking into account the system's critical instant. On the other hand, the LHA model allows parametric schedulability analysis on scenarios that the analytical approach does not apply.

In Chapter 4, the LHA is utilised to model a periodic server, which is a very classic abstraction in real-time scheduling theory. As a result, the LHA approach brings more precise schedulability results; due to the expressness, the LHA model can deal with different task characterisations.

The multiprocessor global scheduling problem has been extensively studied in this work. At first, a LHA model is built to solve the exact Global Fixed Priority (G-FP) schedulability analysis. Because of the high complexity, an exact analysis reaches state explosion easily. Then, the heuristic from real-time scheduling helps and a pre-order relation is defined to mitigate the state explosion problem. Meanwhile, based on the LHA model, a bounded time interval for deciding a sporadic task's schedulability under G-FP scheduling, however, this result itself does not depend on the underlying LHA model and it advances people's understanding on the G-FP scheduling of sporadic tasks.

Even with the proposed pre-order relation, the exact analysis can only deal with rather small systems. Still, the analytical approach with over-approximate schedulability results has its critical role. Thus, more precise schedulability tests for multiprocessor global FP and global EDF systems have been developed in Chapter 7 and Chapter 8.

One key difference between the exact G-FP test in LHA and the analytical tests upon multiprocessor global scheduling is that the LHA based test is in continuous time and the latter often assumes the discrete time schedule. When reasoning such a mismatch, the robustness problem for schedulability analysis in discrete time domain becomes a concern. This motivates the work in Chapter 9 for the task parameter scalability problem. As a result, the exact test in discrete time domain for some scheduling policies becomes problematic, and the G-FP policy belongs to this class of scheduling algorithms. Moreover, state-of-the-art analytical tests for G-FP become unsafe to use and a new methodology for safe G-FP schedulability analysis is proposed.

Overall, the analytical approach for schedulability analysis focuses on special cases of a system, for example, the worst-case scenario; in the model-based approach, each state in the state space represents a specific scenario of the system. In conclusion, the most important experience learnt from a series of works in this thesis is that the integration (also the collision) of different solution techniques does bring people new perspectives to analyse a real-time system.

Acknowledgments

Three years ago, when I started my PhD work, my naive plan was to become famous in some specific field. While this is not really a wise motivation, I do not regret my choice of joining the PhD program.

I am deeply appreciative of my supervisor Giuseppe Lipari for his guidance, respect, tolerance, kindness and support on me during my PhD study. He deserves all these words, and I believe I learnt many important things from him besides being professional.

I am lucky to work with many brilliant colleagues. I owe my thanks to Marco Di Natale, who has helped me in different circumstances since I was a graduate student. I would like to thank Giorgio Buttazzo. Under his lead, the RETIS lab is the perfect place for conducting research work on real-time systems and I feel proud being a member of it. Thanks to captain Enrico Bini of our carcassonne team.

I am grateful to Marko Bertogna and Gilles Geeraerts for revising my PhD thesis. Moreover, their previous work is the base for my results in the thesis.

Thank Laurent Fribourg for inviting me to visit LSV, where I began the adventure in the wonderland of formal methods. Special thanks to Étienne André, for our interesting discussions and amazing works together on parametric Timed Automata.

Thank Claudio Manfroni and Valentina Venuti, and they have been always available to help on administrative stuff. Thank all the past and current members in the lab, my collaborators and my friends.

Finally, thanks go to my family for the love, trust and support!

Bibliography

[AB98] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, Spain, December 1998.

[ABR+93] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy J Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. *Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems*. Springer, 1993.

[AD94] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[AF10] Étienne André and Laurent Fribourg. Behavioral cartography of timed automata. In *Reachability Problems*, pages 76–90. Springer, 2010.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In Dimitra Giannakopoulou and Dominique Méry, editors, *Proc. 18th International Symposium Formal Methods (FM'12)*, volume 7436 of *LNCS*, pages 33–36, Paris, France, August 2012. Springer.

[AFS13] Étienne André, Laurent Fribourg, and Romain Soulat. Merge and conquer: State merging in parametric timed automata. In *ATVA*, volume 8172 of *Lecture Notes in Computer Science*, pages 381–396. Springer, 2013.

[AHV93] Rajeev Alur, Thomas A Henzinger, and Moshe Y Vardi. Parametric real-time reasoning. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 592–601. ACM, 1993.

[ALNS15] Étienne André, Giuseppe Lipari, Hoang Gia Nguyen, and Youcheng Sun. Reachability preservation based parameter synthesis for timed automata. In *NASA Formal Methods*, pages 50–65. Springer, 2015.

[AM02] Yasmina Abdeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 113–126. Springer, 2002.

[APN11] Mikael Asberg, Paul Pettersson, and Thomas Nolte. Modelling, verification and synthesis of two-tier hierarchical fixed-priority preemptive scheduling. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 172–181. IEEE, 2011.

[ARI96] ARINC. *ARINC 653: Avionics Application Software Standard Interface (Draft 15)*. Airlines Electronic Engineering Committee (AEEC), June 1996.

[AS13] Étienne André and Romain Soulat. *The Inverse Method*. ISTE Ltd and John Wiley & Sons Inc., 2013.

[Bak03] Theodore Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. *IEEE Real-Time Systems Symposium (RTSS)*, 2003.

[Bar07] Sanjoy K. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 119–128, 2007.

[BB06] Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pages 159–168. IEEE, 2006.

[BB09a] Theodore Baker and Sanjoy Baruah. An analysis of global EDF schedulability for arbitrary-deadline sporadic task systems. *Real-Time Systems*, 43(1):3–24, 2009.

[BB09b] Theodore Baker and Sanjoy K Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*, pages 141–150. IEEE, 2009.

[BB11] Marko Bertogna and Sanjoy Baruah. Tests for global EDF schedulability analysis. *Journal of Systems Architecture*, 57(5):487–497, 2011. Special Issue on Multiprocessor Real-time Scheduling.

[BBMSS09] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. Implementation of a speedup-optimal global EDF schedulability test. In *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*, pages 259–268. IEEE, 2009.

[BC07a] Theodore P Baker and Michele Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *Principles of Distributed Systems*, pages 62–75. Springer, 2007.

[BC07b] Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 149–160. IEEE, 2007.

[BCL05] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pages 209–218. IEEE, 2005.

[BDG+11] T Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J.F. Raskin, and J. Worrell. On reachability for hybrid automata over bounded time. In *Proc. 38th International Colloquium on Automata, Languages and Programming (ICALP'11)*, volume 6756 of *LNCS*, pages 416–427. Springer, 2011.

[BDNB08] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, 39(1-3):5–30, 2008.

[BFSV04] Giacomo Bucci, Andrea Fedeli, Luigi Sassoli, and Enrico Vicario. Timed state space analysis of real-time preemptive systems. *Software Engineering, IEEE Transactions on*, 30(2):97–111, 2004.

[BMR90] Sanjoy K Baruah, Aloysius K Mok, and Louis E Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190. IEEE, 1990.

[BMS12] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 63(4):763–780, 2012.

[BRH90] Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time systems*, 2(4):301–324, 1990.

[But11] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[CG06] Liliana Cucu and Joël Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on*, pages 397–404. IEEE, 2006.

[CG07] Liliana Cucu and Joël Goossens. Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1635–1640. EDA Consortium, 2007.

[CGG11] Liliana Cucu-Grosjean and Joël Goossens. Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms. *Journal of systems architecture*, 57(5):561–569, 2011.

[CL00] Franck Cassez and Kim Larsen. The impressive power of stopwatches. In *CONCUR 2000Concurrency Theory*, pages 138–152. Springer, 2000.

[CLPV11] Laura Carnevali, Giuseppe Lipari, Alessandro Pinzuti, and Enrico Vicario. A formal approach to design and verification of two-level hierarchical scheduling systems. In Alexander Romanovsky and Tullio Vardanega, editors, *Proc. 16th Ada-Europe International Conference Reliable Software Technologies (Ada-Europe'11)*, volume 6652 of *LNCS*, pages 118–131. Springer, 2011.

[CPR08] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *Real-Time Systems Symposium, 2008*, pages 80–89. IEEE, 2008.

[CPV13] Laura Carnevali, Alessandro Pinzuti, and Enrico Vicario. Compositional verification for hierarchical scheduling of real-time systems. *IEEE Trans. on Software Engineering*, 39(5):638–657, 2013.

[DB05] R.I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proc. 26th IEEE Real-Time Systems Symposium (RTSS'05)*, pages 10 pp.–398, 2005.

[DB11] Robert Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2011.

[Dod06a] RB Dodd. An analysis of task-scheduling for a generic avionics mission computer. Technical report, DTIC Document, 2006.

[Dod06b] RB Dodd. Coloured petri net modelling of a generic avionics mission computer. Technical report, DTIC Document, 2006.

[ESD10] Paul Emberson, Roger Stafford, and Robert Davis. Techniques for the synthesis of multiprocessor tasksets. In *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, 2010.

[FJK08] Goran Frehse, Sumit Kumar Jha, and Bruce H Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *Hybrid Systems: Computation and Control*, pages 187–200. Springer, 2008.

[FLMS12] Laurent Fribourg, David Lesens, Pierre Moro, and Romain Soulat. Robustness analysis for scheduling problems using the inverse method. In Mark Reynolds, Paolo Terenziani, and Ben Moszkowski, editors, *Proc. 19th International Symposium Temporal Representation and Reasoning (TIME'12)*, pages 73–80, Leicester, UK, September 2012. IEEE Comp. Soc. Press.

[flo] Introduction to the rounding and congruence functions. Web page: http://functions.wolfram.com/IntegerFunctions/Ceiling.

[FM02] Xiang Feng and Aloysius K. Mok. A model of hierarchical real-time virtual resources. In *Proc. 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, pages 26–35, Austin, TX USA, December 2002.

[FMPY03] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis using two clocks. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 224–239. Springer, 2003.

[FPY02] Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 67–82. Springer, 2002.

[GGCG13] Emmanuel Grolleau, Joël Goossens, and Liliana Cucu-Grosjean. On the periodic behavior of real-time schedulers on identical multiprocessor platforms. *arXiv preprint arXiv:1305.3849*, 2013.

[GGD+07] Nan Guan, Zonghua Gu, Qingxu Deng, Shuaihong Gao, and Ge Yu. Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In *Software Technologies for Embedded and Ubiquitous Systems*, pages 263–272. Springer, 2007.

[GGL+08] Nan Guan, Zonghua Gu, Mingsong Lv, Qingxu Deng, and Ge Yu. Schedulability analysis of global fixed-priority or EDF multiprocessor scheduling with symbolic model-checking. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 556–560. IEEE, 2008.

[GGL13] Gilles Geeraerts, Joël Goossens, and Markus Lindstrm. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm. *Real-Time Systems*, 49(2):171–218, 2013.

[GGS14] Gilles Geeraerts, Joël Goossens, and Amélie Stainer. Synthesising succinct strategies in safety and reachability games. In *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, pages 98–111, 2014.

[GSYY09] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 387–397. IEEE, 2009.

[GY14] Nan Guan and Wang Yi. General and efficient response time analysis for EDF scheduling. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6, 2014.

[Has12] Gomaa Hassan. *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge University Press, 2012.

[HPR97] Nicolas Halbwachs, Yann-Erick Proy, and Patrick Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.

[LA13] Cong Liu and James H Anderson. Suspension-aware analysis for hard real-time multiprocessor scheduling. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 271–281. IEEE, 2013.

[LAL+14] Shang-Wei Lin, Etienne Andre, Yang Liu, Jun Sun, and Jin Song Dong. Learning assumptions for compositionalverification of timed systems. *IEEE Transactions on Software Engineering*, 2(40):137–153, 2014.

[LB04] Giuseppe Lipari and Enrico Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2), 2004.

[Lee14] Jinkyu Lee. Time-reversibility of schedulability tests. In *Real-Time Systems Symposium (RTSS), 2014 IEEE 35th*. IEEE, 2014.

[LFCL12] Juri Lelli, Dario Faggioli, Tommaso Cucinotta, and Giuseppe Lipari. An experimental comparison of different real-time schedulers on multicore systems. *Journal of Systems and Software*, 85(10):2405–2416, 2012.

[Liu00] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[LL73] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[LPPR13] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, and Yusi Ramadian. Timed-automata based schedulability analysis for distributed firm real-time systems: a case study. *International Journal on Software Tools for Technology Transfer*, 15(3):211–228, 2013.

[LPT13] Kai Lampka, Simon Perathoner, and Lothar Thiele. Component-based system design: analytic real-time interfaces for state-based component implementations. *International Journal on Software Tools for Technology Transfer*, 15(3):155–170, 2013.

[LS13] Jinkyu Lee and Insik Shin. Limited carry-in technique for real-time multi-core scheduling. *Journal of Systems Architecture*, 59(7):372–375, 2013.

[LSSE15] Jinkyu Lee, Kang G. Shin, Insik Shin, and Arvind Easwaran. Composition of schedulability analyses for real-time multiprocessor systems. *IEEE Trans. Computers*, 64(4):941–954, 2015.

[LW82] Joseph Y-T Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250, 1982.

[Min01] Antoine Miné. A new numerical abstract domain based on difference-bound matrices. In *Proceedings of the Second Symposium on Programs as Data Objects*, PADO '01, pages 155–172, London, UK, UK, 2001. Springer-Verlag.

[Min06] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[SLa] Youcheng Sun and Giuseppe Lipari. A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor global fixed-priority scheduling. *Real-Time Systems*.

[SLb] Youcheng Sun and Giuseppe Lipari. The task parameter scalability problem in real-time systems.

[SL03] Insik Shih and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proc. 24th IEEE Real-Time Systems Symposium (RTSS'03)*, pages 2–13, Cancun, Mexico, December 2003.

[SL08] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):30, 2008.

[SL14a] Youcheng Sun and Giuseppe Lipari. FOrmal Real-Time Scheduler (FORTS). Web page: `https://github.com/glipari/forts`, January 2014.

[SL14b] Youcheng Sun and Giuseppe Lipari. A weak simulation relation for real-time schedulability analysis of global fixed priority scheduling using linear hybrid automata. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, page 35. ACM, 2014.

[SL15] Youcheng Sun and Giuseppe Lipari. Response time analysis with limited carry-in for global earliest deadline first scheduling. In *Real-Time Systems Symposium (RTSS), 2015 IEEE 36th*. IEEE, 2015.

[SLAF14] Youcheng Sun, Giuseppe Lipari, Étienne André, and Laurent Fribourg. Toward parametric timed interfaces for real-time components. In *Proceedings 1st International Workshop on Synthesis of Continuous Parameters (SynCoP)*, pages 49–64, Grenoble, France, 6th April 2014.

[SLGY14] Youcheng Sun, Giuseppe Lipari, Nan Guan, and Wang Yi. Improving the response time analysis of global fixed-priority multiprocessor scheduling. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–9. IEEE, 2014.

[SLS+14] Youcheng Sun, Giuseppe Lipari, Romain Soulat, Laurent Fribourg, and Nicolas Markey. Component-based analysis of hierarchical scheduling using linear hybrid automata. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10. IEEE, 2014.

[Spe91] ARINC Specification. 651: Design guidance for integrated modular avionics. *Aeronautical Radio, Inc, Annapolis, MD*, 1991.

[Spu96] Marco Spuri. Analysis of deadline scheduled real-time systems. 1996.

[SSL+14] Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. Parametric schedulability analysis of fixed priority real-time distributed systems. In *Formal Techniques for Safety-Critical Systems*, pages 212–228. Springer, 2014.

[TCN00] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104. IEEE, 2000.

[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H Roux. Parametric model-checking of stopwatch petri nets. *J. UCS*, 15(17):3273–3304, 2009.

[WW07] Christopher B Watkins and Randy Walter. Transitioning from federated avionics architectures to integrated modular avionics. In *Digital Avionics Systems Conference, 2007. DASC'07. IEEE/AIAA 26th*, pages 2–A. IEEE, 2007.