FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Schedulability Analysis of Multiprocessor Real-time Systems Using Pruning

Artem Burmyakov

Programa Doutoral em Engenharia Electrotécnica e de Computadores

Supervisor: Prof. Dr. Eduardo Manuel Medicis Tovar

Co-supervisor: Dr. Vincent Nélis

July 25, 2016

© Artem Burmyakov, 2016

Schedulability Analysis of Multiprocessor Real-time Systems Using Pruning

Artem Burmyakov

Programa Doutoral em Engenharia Electrotécnica e de Computadores

Approved by:

President: Dr. Jose Alfredo Ribeiro da Silva Matos
External referee: Dr. Giuseppe Lipari
External referee: Dr. Enrico Bini
FEUP referee: Dr. Luis Miguel Pinho de Almeida
FEUP referee: Dr. Mario Jorge Rodrigues de Sousa
Supervisor: Dr. Eduardo Manuel Medicis Tovar

July 25, 2016

Abstract

We address a range of schedulability analysis problems for multiprocessor real-time systems. As most of these problems are NP-complete and no efficient solutions have been so far proposed, we tackle these problems by employing mathematical optimization combined with the dynamic pruning of the solution search space. We show that such an approach results in solving algorithms of a significantly lower time and space complexity compared to other state-of-the-art solutions. In the best case, the complexity can be reduced from exponential down to just linear.

We use the pruning approach to solve a range of schedulability analysis problems. We first derive an exact schedulability test for sporadic real-time tasks, scheduled by Global Fixed Priority (GFP). Our test is faster and less memory consuming, compared to all previously known exact tests. We achieve such results by pruning the state space through employing such constraints as i) a sufficient schedulability condition, ii) the length bound for the longest release sequence to be considered, iii) critical job release instants, iv) an optimized clock transition between checked system states, as well as others.

We also extend the test to continuous-time schedulers (e.g. an event-driven scheduler, which is opposed to a tick-driven scheduler), discarding the assimption of discrete time and integer task parameters, by employing linear programming methods combined with the search space pruning, instead of naive enumeration methods.

Another considered problem is the schedulability analysis of compositional multiprocessor real-time systems. Our solution is based on solving a set of mixed-integer non-convex optimization problems. Due to their high complexity, no existing solver could efficiently determine a solution. To make solving possible, we first prune the solution search space, exploring the theoretical insides of the scheduling problem, and then employ an adequate optimization solver over a reduced search space, so that it quickly finds an optimal solution. The resulted search space is so narrow, that approximate solvers perform almost as good as exact ones: in most cases an approximate solver finds a true global optimum, but significantly faster compared to an exact solver.

All solutions have been implemented in C++ and Matlab environments, and they are publicly available.

Resumo

Nesta dissertação focamos um conjunto de problemas de análise de escalonamento para sistemas de tempo-real baseados em plataformas multi-processador. Uma vez que estes problemas apresentam uma complexidade NP-complete e considerando a não existência de soluções verdadeiramente eficientes, abordamos estes problemas através de uma estratégia de optimização matemática combinada com uma eliminação progressiva (dynamic pruning) do conjunto solução. Provamos que esta estratégia permite reduzir a complexidade espacial e temporal dos algoritmos quando comparada com outras alternativas recentes. Deste facto resulta que, no melhor caso, a complexidade pode ser decrementada de um grau exponencial até um linear.

Usamos a estratégia de eliminação progressiva para resolver uma série de problemas de análise de escalonamento, nomeadamente, para conseguir um teste de escalonabilidade exacto para tarefas esporádicas e de tempo-real, escalonadas com Global Fixed Prioriy (GFP). O teste de escalonabilidade conseguido é mais rápido e mais eficiente em termos de memória utilizada quando comparado com todos os outros testes já propostos. Este resultado é conseguido através do processo de eliminação progressiva do conjunto de estados, considerando os seguintes requisitos: i) uma condição de escalonabilidade suficiente; ii) o limite para o maior período de sequência de tarefas a ser considerado; iii) os instantes críticos de início de cada tarefa; iv) uma transição de relógio optimizada entre os estados verificados do sistema e restantes estados.

Nesta dissertação também estendemos o teste referido acima para escalonadores de tempo contínuo (um escalonador baseado em eventos em vez de instantes temporais), descartando a premissa da necessidade de parâmetros de tarefas inteiros e de tempo discreto, utilizando para isso métodos de programação linear combinados com a estratégia de eliminação progressiva já referida, em vez de métodos de enumeração.

A análise de escalonamento de sistemas de tempo-real, multi-processador e compostos foi também estudada nesta tese. A solução proposta é baseada na resolução de um conjunto de problemas de optimização não convexos. Devido à grande complexidade inerente a este tipo de sistemas, não foi possível até hoje determinar uma solução de uma forma eficiente. Nesse sentido, propomos uma abordagem que começa por eliminar progressivamente o conjunto de soluções, explorando as premissas teóricas deste tipo de problemas, e depois procurando a sua solução já com um conjunto de soluções menor. Após este processo de eliminação, o conjunto de soluções é tão diminuto que a aplicação de métodos aproximados quer exactos resulta de forma extremamente semelhante em termos de eficiência.

Todas as propostas foram implementadas em C++ e em MATLAB e encontram-se disponíveis.

Contents

1 Introduction to Real-time Scheduling **1**
1.1 Global FP and EDF schedulers . . . 2
1.2 Schedulability analysis . . . 3
1.3 Discrete and continuous-time scheduling . . . 4
1.4 Open problems in real-time scheduling . . . 4
1.5 Search space pruning . . . 5
1.6 Thesis . . . 6
1.7 Contributions . . . 6
1.8 Outline . . . 8

2 An Exact Schedulability Test for Global Fixed Priorities (GFP) **9**
2.1 Motivation . . . 9
2.2 Related works . . . 9
2.3 Definitions . . . 10
2.4 Background on exact schedulability tests . . . 12
2.5 An exact schedulability test using state space pruning . . . 15
2.5.1 Pruning constraints . . . 16
2.5.1.1 Job interference . . . 16
2.5.1.2 Sufficient schedulability condition . . . 19
2.5.1.3 Critical release instant . . . 22
2.5.1.4 Optimized clock transition . . . 25
2.5.2 Test procedure . . . 27
2.5.3 Evaluation . . . 27
2.5.3.1 Task set generation . . . 27
2.5.3.2 Experiments: Runtime reduction . . . 29
2.5.3.3 Experiments: Comparison of exact and sufficient tests . . . 29
2.6 An exact schedulability test for continuous-time schedulers using linear programming (LP) . . . 31
2.6.1 Schedule model . . . 31
2.6.2 Procedure of the test . . . 33
2.7 An exact schedulability test using constraint programming (CP) . . . 34
2.8 Summary . . . 38

3 Schedulability Analysis of Compositional Real-time Systems **39**
3.1 Motivation . . . 39
3.2 Contributions . . . 40
3.3 Related works . . . 40
3.4 Background on compositional scheduling . . . 42

3.4.1 The multiprocessor bandwidth interface (MBI) . . . 42
3.4.2 The multiprocessor periodic resource model (MPR) . . . 43
3.4.3 Comparison of MBI and MPR resource interfaces . . . 44
3.4.4 The parallel supply function (PSF) . . . 45
3.5 The generalized multiprocessor periodic resource (GMPR) model . . . 45
3.5.1 Definition . . . 46
3.5.2 Parallel supply functions of GMPR . . . 47
3.5.3 GMPR supply bounds . . . 50
3.5.4 Supply functions for GMPR with different processor periods . . . 52
3.6 Schedulability over GMPR . . . 53
3.6.1 Simplified schedulability test . . . 54
3.7 Computation of GMPR using pruning . . . 55
3.7.1 Minimal necessary parallelism for GMPR . . . 56
3.7.2 Computation of GMPR aggregated resource . . . 57
3.7.3 Pruned search space for GMPR resource . . . 58
3.7.4 Computation of GMPR resources at lower parallelisms . . . 60
3.7.5 Algorithm to compute GMPR . . . 61
3.8 Scheduling GMPR interfaces . . . 62
3.9 Evaluation . . . 62
3.9.1 Task set generation . . . 62
3.9.2 Experiments: Resource gain . . . 63
3.9.3 Experiments: Runtime analysis . . . 66
3.10 Summary . . . 68

4 Conclusion **69**

A Proofs **71**
A.1 Proof of Theorem 1 . . . 71
A.2 Proof of Corollary 2 . . . 76
A.3 Proof of Theorem 2 . . . 81

References **85**

Publications **89**

Acknowledgements **91**

List of Figures

1.1 Release scenario for a periodic task . . . 2
1.2 Release scenario for a sporadic task . . . 2
1.3 Graphical notation for a resource schedule (used throughout the manuscript) . . . 2
1.4 GFP schedule . . . 3
1.5 GEDF schedule . . . 3
1.6 A schematic illustration of a search space pruning for an exact schedulability test 6

2.1 GFP schedule . . . 12
2.2 State transition graph . . . 15
2.3 Schedule transformation for Theorem 1 by excluding non-interfering jobs . . . 17
2.4 Pruned state transition graph by constraint (2.14) of interfering jobs. . . . 19
2.5 The performance comparison of pruning constraints . . . 20
2.6 The longest resource schedule for consideration in the schedulability test . . . 21
2.7 Maximized job interference . . . 22
2.8 Critical release instants . . . 24
2.9 Optimized clock transition: cases . . . 25
2.10 Evaluation: runtime of exact tests . . . 30
2.11 Evaluation: performance of Guan's sufficient test . . . 30
2.12 Evaluation: comparison of the performance of pruning constraints . . . 31
2.13 The lower and the upper bounds for the number of scheduler invocations . . . 36

3.1 MBI: the worst-case resource pattern . . . 42
3.2 MPR: the worst-case resource pattern . . . 43
3.3 Comparison of MBI and MPR resource interfaces . . . 44
3.4 An example of a resource allocation scenario for PSF computation . . . 45
3.5 GMPR definition: the graphical interpretation . . . 47
3.6 GMPR: the worst-case resource pattern by Burmyakov et al. (2012) . . . 48
3.7 Properties of GMPR supply function . . . 49
3.8 GMPR: the worst-case resource patterns . . . 49
3.9 An example of PSF computation for GMPR . . . 51
3.10 The lower and the upper bounds for GMPR supply . . . 51
3.11 The lower bound for GMPR supply . . . 52
3.12 The worst-case resource patterns for GMPR with different processor periods . . . 53
3.13 Graphical interpretation of the PSF-based schedulability test . . . 55
3.14 Evaluation: GMPR gain for interface period . . . 64
3.15 Evaluation: GMPR gain for maximum task utilization . . . 64
3.16 Evaluation: GMPR gain for ratio of task periods . . . 65
3.17 Evaluation: GMPR gain for task set utilization . . . 65

3.18 Evaluation: GMPR gain for the parallelism of an interface 66
3.19 Evaluation: comparison of computation time for GMPR and MPR 67

A.1 An upper bound on the length of the longest schedule to be examined by the schedulability test 77
A.2 A tighter upper bound on the length of the longest schedule to be examined by the schedulability test 78
A.3 A sequence of interfering jobs: cases 78
A.4 Theorem 2: GMPR worst-case resource patterns 81
A.5 Proof of Theorem 2: case 1(a) 82
A.6 Proof of Theorem 2: case 1(b) 83
A.7 Proof of Theorem 2: case 2 83
A.8 Proof of Theorem 2: comparison of replenishment cycle counters 84

Abbreviations and Main Symbols

CP	Constraint Programming
GEDF	Global Earliest Deadline First
GFP	Global Fixed Priority
GMPR	Generalized Multiprocessor Periodic Resource
LP	Linear Programming
MBI	Multiprocessor Bandwidth Interface
MPR	Multiprocessor Periodic Resource
PSF	Parallel Supply Function

$\mathbb{N}_0$	The set of numbers $0, 1, 2, \dots$
$\mathbb{R}_0^+$	The set of all non-negative real values, including 0
$\underline{x}(t), \bar{x}(t)$	The lower and the upper bounds for function $x(t)$
$(x)_0$	Returns x, if $x > 0$, and 0 otherwise
$\lfloor x \rfloor_0$	Returns $\lfloor x \rfloor$, if $x > 0$, and 0 otherwise
$\lvert\mathbb{G}\rvert$	The number of elements in set $\mathbb{G}$
m	The number of processors; the concurrency of the interface
t	Time instant
Δt	Clock transition between consecutive system states in a state transition graph; the length of a time interval

Task set

$\mathscr{T} = \{\tau_1, \dots, \tau_n\}$	The set of real-time periodic / sporadic tasks
$\tau_i = (C_i, D_i, P_i)$	Real-time task
C_i, D_i, P_i	Execution time, relative deadline, and period of τ_i, respectively
τ_k	Task under schedulability analysis
$U_{\mathscr{T}}$	Total utilization of a task set $\mathscr{T}$, computed by $U_{\mathscr{T}} = \sum_{i=1}^{n} C_i/P_i$
$U_{\max}$	The maximum individual task utilization, $U_{\max} = \max_{i=1,\dots,n} C_i/P_i$
$P_{\min}$, $P_{\max}$	The minimum and the maximum task periods in a task set $\mathscr{T}$

Resource schedule

$\bar{t}$	A sufficient length of a time window for an exact schedulability test
R	Release sequence
F	Finishing sequence
S	Resource schedule
Q	The set of pending jobs

$r_i(t)$, $r_{i,j}$	Indicator function of job releases by a task τ_i
$f_i(t)$, $f_{i,j}$	Indicator function of finishing jobs for a task τ_i
$s_i(t)$, $s_{i,j}$	Supply indicator function allocated to a task τ_i
$q_i(t)$, $q_{i,j}$	Indicator function of a pending job for a task τ_i
$J_{i,\ell}$	The ℓ-th job released by a task τ_i
$\overline{W}_i$	The maximum generated workload by a task τ_i over a certain time interval
$\overline{W}$	The maximum aggregated workload for tasks $\tau_1, \ldots, \tau_{k-1}$
W_i	The maximum interfering workload experienced by a task τ_i
W_{ji}	The maximum interfering workload caused by a task τ_j on τ_i

State transition graph

$g = (c_i, d_i, p_i)_{i=1}^{n}$	System state in a state transition graph, at time t
c_i, d_i, p_i	Respectively, remaining execution time, time until a deadline, time until the next earliest possible release by a task τ_i
$g' = (c'_i, d'_i, p'_i)$	A successor state for a state g
G'	The set of successor states for state g
V	The set of visited system states by a traversing algorithm
G	The set of states for further examination

Compositional scheduling

Π	Interface period
ω	The resource bandwidth of MBI interface
Θ	The aggregated resource of MPR interface
Θ_k	The GMPR aggregated resource provided by at most k concurrency levels
$Y_k(t)$	PSF function
$\mathsf{supply}_k(t)$	The supply function of GMPR
$s_k(t)$	The auxiliary supply function of GMPR
$\mathbb{D}_\Theta$, S_Θ	The definition space and the pruned search space for GMPR resources

Chapter 1

Introduction to Real-time Scheduling

The key property of a real-time system is time predictability: the operations in such a system must be performed within a strictly defined time. A correct behaviour of a real-time system depends not only on the logical correctness of performed actions, but also on the time at which they are performed. Real-time systems are present in many application areas, such as industial control, avionics, telecommunications, stock trading platforms, and others (Davis and Burns, 2011a).

A real-time system is defined by its workload, an execution platform, and a scheduling algorithm.

A real-time workload is modeled by a set of periodic or sporadic tasks $\mathscr{T} = \{\tau_1, \ldots, \tau_n\}$, wherein each task $\tau_i = (C_i, D_i, P_i)$ is characterized by an execution time C_i, a relative deadline D_i, and a period P_i for a periodic task τ_i, or a minimum interarrival time P_i between jobs for a sporadic task τ_i (Liu and Layland, 1973).

Tasks in $\mathscr{T}$ release an infinite sequence of jobs. The first job for each task $\tau_i \in \mathscr{T}$ can be released at an arbitrary time. (We call such a time instant as the activation time of τ_i.) If task τ_i is periodic, then it releases another job at every P_i time units starting from the release instant of its first job. If instead τ_i is sporadic, then it releases another job at an arbitrary time, but not earlier than P_i time units after the release time of its previous job.

Figs. 1.1 and 1.2 provide the examples of release scenarios for task $\tau = (2, 4, 5)$, with Fig. 1.1 assuming that task τ is periodic, and Fig. 1.2 assuming that τ is sporadic. Task τ is set to release its first job at time $t = 0$.

The graphical notation presented in Fig. 1.3 is used throughout this manuscript. An upward arrow denotes the release of a job, and a downward arrow denotes the completion of a job. A dashed line denotes the deadline of a job, and a dashed upward arrow denotes the earliest possible release of the next job, in case of a sporadic task. Finally, a grey block denotes the amount of resource allocated to a job (a resource unit corresponds to one CPU time unit).

A job of task τ_i is completed after receiving C_i resource units. A deadline of a job is D_i time units after its release time; each job must be completed by its deadline, otherwise a job is said to miss its deadline.

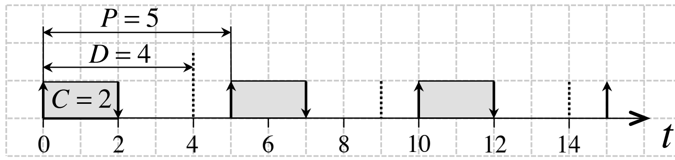


Figure 1.1: Release scenario for a periodic task $\tau = (2,4,5)$

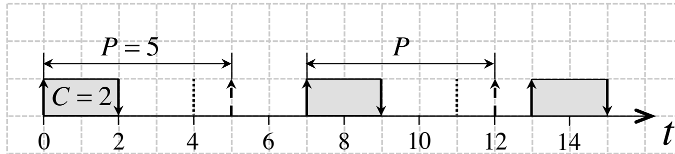


Figure 1.2: Release scenario for a sporadic task $\tau = (2,4,5)$

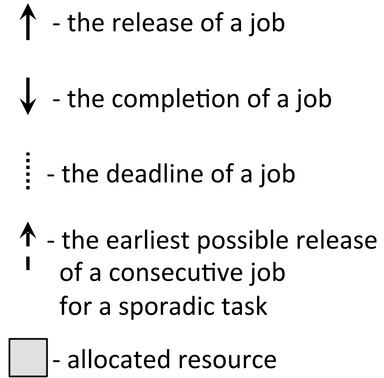


Figure 1.3: Graphical notation for a resource schedule (used throughout the manuscript)

We assume that tasks have constrained deadlines, such that $D_i \leq P_i$. We also restrict jobs to execute upon one processor at a time; no concurrent execution of the same job is allowed.

An execution platform is modeled by the number of processors m available to execute jobs. Each processor has an equal execution speed. We consider a multiprocessor system with $m > 1$.

A scheduling algorithm is used to allocate processors to pending jobs, with a job said pending if it is waiting for execution. When the number of pending jobs exceeds the number of processors, a scheduler prioritises pending jobs by a certain rule, and allocates processors to the m highest priority jobs. Many scheduling algorithms are available, such as fixed priority (FP) or the earliest deadline first (EDF).

A comprehensive taxonomy of real-time systems is provided by Davis and Burns (2011a).

1.1 Global FP and EDF schedulers

We consider two scheduling algorithms, global fixed priorities (GFP) and global earliest deadline first (GEDF). Term "global" means that jobs are allowed to migrate between various processors. A job can preempt the execution of other lower priority jobs. We consider no migration or preemption costs. Unless stated otherwise, we consider discrete-time scheduling, that is scheduling decisions are taken at discrete time instants $t \in \mathbb{N}_0$ with $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$. We next describe how job priorities are determined in case of GFP and GEDF.

For GFP, each task has a pre-defined fixed priority. We assume that tasks in $\mathscr{T}$ are sorted by decreasing priorities, meaning that task τ_i has a higher priority than τ_{i+1}. At any time instant, GFP allocates processors to the m highest priority jobs, while the remaining jobs are suspended from execution.

In Fig. 1.4 we provide an example of a GFP schedule for a sporadic $\mathscr{T}$ with parameters reported in Table 1.1, which is scheduled upon $m = 2$ processors. Note that at time $t = 2$, the number

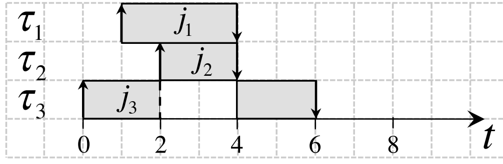


Figure 1.4: GFP schedule

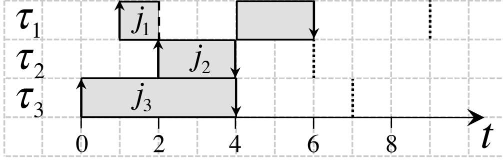


Figure 1.5: GEDF schedule

i	C_i	P_i	D_i
1	3	8	8
2	2	4	4
3	4	7	7

Table 1.1: Task set parameters (sporadic)

of pending jobs exceeds the number of processors (3 pending jobs, but only two processors); the processors are allocated to jobs of higher priority tasks τ_1 and τ_2.

Unlike GFP with pre-defined fixed priorities for tasks, GEDF prioritises jobs dynamically over time, by assigning higher priorities to jobs with earlier deadlines. Fig. 1.5 provides an example of a GEDF schedule, for $\mathscr{T}$ with parameters listed in Table 1.1, scheduled upon 2 processors. Observe that at time $t = 2$, when the number of pending jobs exceeds the number of processors, GEDF allocates processors to tasks τ_2 and τ_3, instead of tasks τ_1 and τ_2, as in case of GFP depicted in Fig. 1.4.

1.2 Schedulability analysis

For a real-time system, all jobs should meet their deadlines. A system is said schedulable, if all job deadlines are guaranteed to be met, for any legal scenario of job releases; otherwise a system is said unschedulable.

To check if a system is schedulable, a schedulability test is used. Various schedulability tests exist, which can be exact, sufficient, or necessary, depending on their accuracy in provisioning the resource demand by a system.

An exact test correctly defines all schedulable and unschedulable systems, meaning that a system $\mathscr{T}$ is schedulable if and only if it satisfies such a test. However, an exact schedulability test for a multiprocessor system is considered to be an NP-hard problem. No efficient exact test has been derived so far, which can be applied to systems with realistic task parameters (see evaluation in Section 2.5.3). Thus, sufficient and necessary tests have been also introduced.

A sufficient schedulability test overestimates the system demand for resource. If system $\mathscr{T}$ meets a sufficient test, then such $\mathscr{T}$ is indeed schedulable. If however $\mathscr{T}$ fails a test, it is not known if $\mathscr{T}$ is schedulable or not. Existing sufficient tests are pessimistic, meaning that many of systems, although violating a sufficient test, are schedulable (see evaluation in Section 2.5.3).

Unlike a sufficient test, a necessary test underestimates the system demand for resource. Each $\mathscr{T}$, violating a necessary test, is unschedulable. If however $\mathscr{T}$ meets a test, it is not known if $\mathscr{T}$ is schedulable or not.

The computational complexity of sufficient and necessary schedulability tests is significantly lower compared to exact tests. In fact, there are various sufficient and necessary tests of polynomial and pseudo-polynomial complexity (in the number of tasks in $\mathscr{T}$), while existing exact tests are strongly exponential.

Necessary and sufficient tests can be used supplimentary to reduce the computation time of an exact test (see Section 2.5.1.2), by pruning the search space for the worst-case scenario of job releases.

Baker and Baruah (2009) have shown that GFP and GEDF are sustainable schedulers, that is the schedulability of sporadic tasks remains in case task execution times are reduced, or/and task periods are increased.

1.3 Discrete and continuous-time scheduling

We distinguish discrete and continuous-time scheduling. Below we describe the key difference between these cases.

If time is discrete, then jobs and scheduling decisions are taken at discrete time instants only, $t \in \mathbb{N}_0$. If instead time is continuous, then jobs can be released at an arbitrary time $t \in \mathbb{R}_0^+$.

Schedulability for discrete time can be checked by a direct enumeration of all possible scenarios for job releases for a system, as the number of feasible release scenarios in this case is finite (over a certain finite time interval). Such an approach however cannot be applied to continuous time, as the number of release scenarios becomes infinite. Note that it is not yet known if the schedulability of a system for discrete time implies the schedulability for continuous time (Baruah and Pruhs, 2010).

On another side, the schedulability analysis is expected to have a higher computational complexity, compared to continuous time, due to a larger number of integer-valued parameters. This is due to the fact that the computational complexity of a problem increases drastically with a number of integer-valued parameters.

Another argument in favor for the continuous-time analysis is that most of real-time operating systems are event-driven, rather than tick-based, e.g. Linux, QNX, RTEMS, what corresponds to a case of continuous-time scheduling.

In this work we consider both cases, discrete and continuous-time scheduling. Unless stated otherwise, time is assumed discrete.

1.4 Open problems in real-time scheduling

There are multiple opened problems in real-time scheduling. One of them is an exact schedulability test for multiprocessor systems. Although pseudopolynomial-time exact schedulability

tests exist for a uniprocessor platform (Liu and Layland, 1973; Joseph and Pandya, 1986; Bini and Buttazzo, 2004), the available exact tests for a multiprocessor platform are highly time and memory-consuming, and they become intractable for systems with realistic task parameters.

The main difficulty in deriving an exact schedulability test for a multiprocessor system is a lack of understanding for the worst-case release scenario for jobs. Most of the exact tests enumerate directly all feasible release scenarios, assuming that time is discrete, and these tests become intractable even for systems with a few tasks only. In fact, schedulability analysis for multiprocessor systems is considered as an NP-hard problem.

Hence, many research works focused on necessary and sufficient tests instead, which however significantly underestimate and overestimate the system demand for processing capacities. For example, a state-of-the-art sufficient test by Guan et al. (2009) fails to identify more than half of schedulable task sets, under certain settings. The pessimism of sufficient tests is expected to increase further, when applied to more general cases, such as compositional real-time systems, systems with precedence constraints between tasks, or DAG tasks.

In Section 2 we propose an exact schedulability test for sporadic tasks, which outperforms the state-of-the-art tests in terms of computation time.

In addition, we address another important problem - schedulability analysis for compositional scheduling. In Section 3, we first derive an exact worst-case resource allocation scenario for a range of multiprocessor resource models, and then we explore it to derive a tigher and faster schedulability test.

1.5 Search space pruning

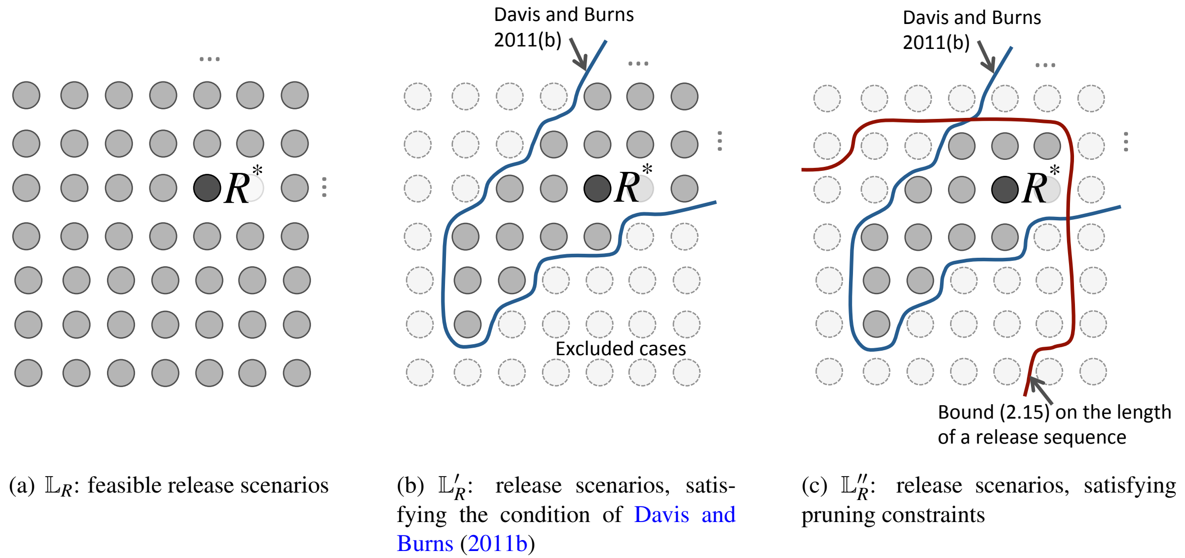
Below we briefly introduce the idea of a search space pruning, which is extensively used in later sections.

Consider a problem of finding an optimal solution over a given set of candidate solutions. An optimal solution is the one satisfying a certain optimality criteria. The aim of pruning is to reduce the computation time for a problem, by shrinking the search space for a solution. Pruning excludes all those candidates from the analysis, which violate a certain necessary condition for an optimal solution. Such a necessary condition for an optimal solution is assumed to be given.

We illustrate graphically the idea of pruning in Fig. 1.6, considering the problem of an exact schedulability test. $\mathscr{T} = \{\tau_1, \ldots, \tau_k\}$ represents a set of sporadic tasks under the analysis, with tasks in $\mathscr{T}$ sorted by decreasing priorities, and scheduled by GFP. We test the schedulability of the lowest-priority task τ_k.

Fig. 1.6(a) represents an infinite set of feasible release scenarios for $\mathscr{T}$, denoted by $\mathbb{L}_R$. In Fig. 1.6(a), each grey circle corresponds to a certain release sequence. Schedulability test aims to determine the existence of such a case $R^* \in \mathbb{L}_R$, that a job of task τ_k misses its deadline.

We next prune $\mathbb{L}_R$. Davis and Burns (2011b) have proved the sufficiency of considering only those release scenarios, wherein task τ_k releases a job simultaneously with at least one more task. Thus, $\mathbb{L}_R$ can be reduced into $\mathbb{L}'_R$, by excluding all release scenarios violating such a condition.



(a) $\mathbb{L}_R$: feasible release scenarios

(b) $\mathbb{L}'_R$: release scenarios, satisfying the condition of Davis and Burns (2011b)

(c) $\mathbb{L}''_R$: release scenarios, satisfying pruning constraints

Figure 1.6: A schematic illustration of a search space pruning for an exact schedulability test

Such $\mathbb{L}'_R$ is depicted in Fig. 1.6(b), where light grey circles with dashed boundaries denote the cases violating the condition of Davis and Burns (2011b). Observe that the worst-case R^* remains in $\mathbb{L}'_R$. Note however that $\mathbb{L}'_R$ remains infinite.

Another pruning condition is derived later in Section 2. According to it, the length of considered release sequences should not exceed the sum of task deadlines. Such a condition allows to reduce $\mathbb{L}'_R$ into $\mathbb{L}''_R$, which is schematically depicted in Fig. 1.6(c). Observe that $\mathbb{L}''_R$ is finite, thanks to such a pruning condition.

To achieve an efficient pruning, the employed pruning constraints should provide a good balance between their computation time and the size reduction of the solution search space. We elaborate the idea of pruning in later sections.

1.6 Thesis

We evaluate the following thesis:

A range of schedulability analysis problems for real-time multiprocessor systems can be efficiently addressed by using a method of a search space pruning.

Our approach is evaluated for an exact schedulability analysis and compositional analysis of real-time multiprocessor systems. In the best case, the exponential runtime complexity of the state-of-the-art solutions is reduced to linear, thanks to pruning.

1.7 Contributions

We derive several methods for the schedulability analysis of multiprocessor real-time systems, by exploring the idea of a search space pruning. Our contributions include:

(a) An exact schedulability test for GFP disrete and continous-time scheduling, which is faster and less memory consuming compared to other state-of-the-art exact tests (Section 2).

(b) A set of necessary conditions for the GFP worst-case execution scenario, which are:

   (a) the maximum length of a schedule, which is bounded by the sum of task deadlines (Section 2.5.1.1);

   (b) conditions for a critical release instant for a job, generalizing the analysis of Davis and Burns (2011b) (Section 2.5.1.3);

   (c) a necessary unschedulability condition, generalizing the analysis of Baruah (2007) (Section 2.5.1.2);

   (d) a job interference condition, requiring each job in a schedule to interfere with some lower priority job (Section 2.5.1.1);

   (e) an optimized clock transition between checked system states (Section 2.5.1.4).

(c) A set of necessary conditions for the worst-case execution scenario independent of a scheduling policy. These conditions include i) the condition for interfering jobs (Section 2.5.1.1), and ii) a necessary unschedulability condition (Section 2.5.1.2).

(d) The analysis of the pessimism of the state-of-the-art sufficient schedulability tests (Section 2.5.3.3).

(e) The formulation of an exact schedulability test for GFP continuous-time scheduling, by using linear programming (LP, Section 2.6) and constraint programming (CP, Section 2.7).

We also derive several methods for compositional scheduling, including:

(a) A generalized multiprocessor periodic interface model (GMPR), which generalizes the other state-of-the-art periodic interface models (Sections 3.5 and 3.5.4).

(b) The analysis of efficiency of the state-of-the-art interface models in abstracting the resource requirements of real-time components (Sections 3.4 and 3.9).

(c) The worst-case resource allocation scenarios for multiprocessor interface models, considering both identical replenishment periods (Section 3.5.2), and arbitrary replenishment periods (Section 3.5.4) for concurrent supplies.

(d) A faster sufficient schedulability test for real-time compositional scheduling, extending the sufficient test by Bini et al. (2009) (Section 3.6).

(e) A method for solving mixed-integer non-convex optimization problems by pruning dynamically a solution search space (Sections 3.7.2 and 3.7.3).

(f) A method to schedule a set of component interfaces at an inter-component level (Section 3.8).

All solutions are implemented in C++ and Matlab environments, which are publicly available.

1.8 Outline

The remainder of the manuscript is organized as follows.

In Section 2 we propose a faster exact schedulability test for sporadic tasks scheduled by GFP. In particular, in Sections 2.1-2.4 we motivate a need for an exact schedulability test, and provide a background information on it. In Section 2.3 we introduce a formal notation to model release and completion sequences for tasks, resource schedules, etc. Our major contributions are described in Section 2.5, where we derive a set of methods for a faster exact schedulability test, including a necessary unschedulability condition, condition for a critical release instant, an optimized clock transition between checked system states, etc. Finally, in Section 2.5.3 we report the evaluation results, showing the runtime reduction for an exact test thanks to our improvements, as well as analyzing the pessimism of the available sufficient tests.

In addition, in Sections 2.6 and 2.7 we propose an exact schedulability test for GFP continuous-time scheduling, by using linear programming methods (LP), and constraint programming methods (CP).

Then, in Section 3, we apply pruning optimization to a different type of scheduling - compositional multiprocessor scheduling. First, in Sections 3.1-3.4 we motivate a need for compositional scheduling, and review the concepts and notations related to our work. In Section 3.5 we introduce a generalized multiprocessor periodic interface model (GMPR), and then, in Sections 3.6 and 3.7, we derive an algorithm to compute GMPR, by using linear programming combined with a dynamic pruning of a solution search space. For completeness, in Section 3.8 we briefly describe a method for scheduling GMPR interfaces between themselves. Finally, in Section 3.9 we evaluate the efficiency of GMPR in abstracting the resource demands, compared to other state-of-the-art models.

The formal proofs for theorems are provided in Appendix A.

Chapter 2

An Exact Schedulability Test for Global Fixed Priorities (GFP)

2.1 Motivation

Guan et al. (2009) have derived a sufficient schedulability test for multiprocessor real-time systems, which in average outperforms all other existing sufficient tests. To understand the need for an exact schedulability test, we thoroughly evaluated the performance of Guan's sufficient test against the exact test of Bonifaci and Marchetti-Spaccamela (2012), and in certain cases, its pessimism exceeds 50%: more than half of those task sets, reported by Guan's test as unschedulable, are in fact schedulable. However, due to high computation time and memory consumption, Bonifaci's exact test becomes intractable even for small systems.

We next propose an improved exact schedulability test, by deriving a set of pruning methods for the state space determined by Bonifaci. We consider a set of sporadic tasks $\mathscr{T} = \{\tau_1, \ldots, \tau_n\}$ scheduled by GFP upon m processors, with tasks in $\mathscr{T}$ sorted by decreasing priorities. For each task $\tau_i = (C_i, D_i, P_i)$, parameters C_i, D_i, P_i are assumed to be integers, as well as deadline D_i is constrained to $D_i \leq P_i$. Scheduling decisions are taken at discrete time instants $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.

In addition, we propose two exact tests for continuous-time scheduling, based on linear programming (LP), and constraint logical programming (CP).

2.2 Related works

The first exact schedulability test has been proposed by Baker and Cirinei (2007), for several discrete-time schedulers. To check whether a given system is schedulable, the authors solve a reachability problem in a finite state transition graph: the algorithm traverses such a graph until it either finds a state with a violated deadline, or all feasible states are confirmed schedulable.

Bonifaci and Marchetti-Spaccamela (2012) improved significantly the test of Baker and Cirinei (2007). They have also refined the complexity bounds for the exact test, showing that it has polynomial space complexity, rather than exponential, as reported in Baker and Cirinei (2007).

An efficient C++ implementation for Bonifaci's test is publicly available[1]. As our work strongly relies on Bonifaci and Marchetti-Spaccamela (2012), Section 2.4 will provide more details on that work.

Another exact test was proposed by Geeraerts et al. (2013), using formal verification methods. Both the test of Bonifaci and Marchetti-Spaccamela (2012) and the test of Geeraerts et al. (2013) apply to most of online discrete-time schedulers, such as GFP and GEDF, and allow tasks with arbitrary deadlines. Sun and Lipari (2014) have instead derived a test specifically for GFP, by using a linear hybrid automaton.

Finally, Guan et al. (2007) proposed a test for strictly periodic tasks with arbitrary offsets, scheduled by GFP, which uses the model-checking techniques from the theory of formal verification. However, in multiprocessor scheduling, the scenario with periodic activations is not the worst-case for sporadic tasks, and an exact test for sporadic tasks must analyze a significantly larger number of legal release sequences.

Our evaluation has shown that the test of Bonifaci and Marchetti-Spaccamela (2012) for GFP is faster, when compared to the exact tests using a timed automaton, that is by Geeraerts et al. (2013), and Sun and Lipari (2014). Such a conclusion is based on comparing running times reported in Geeraerts et al. (2013), Sun and Lipari (2014) against our evaluation reported in Section 2.5.3. The runtime gain of Bonifaci's test increases noticeably for task sets with a larger number of tasks, and a larger range of task periods. For example, while Geeraerts' test is constrained to task periods not exceeding 6-8, Bonifaci's test can deal with larger task periods up to 40.

We have also made some initial evaluation of constraint programming and global optimization methods (such an optimization problem can be formulated through the notation proposed in Section 2.3), but the resulted runtime was much longer than for Bonifaci's test. For these reasons, we have chosen Bonifaci's test as an initial ground to apply our improvements. Anyway, we remark that all runtime reduction techniques derived in this work can be applied to any other existing exact test for GFP.

2.3 Definitions

To represent the possible scenarios of job releases, we define the *release sequence* R as a set of n functions

$$R = \{r_1(t), \ldots, r_n(t) \mid t \in \mathbb{N}_0\},$$

wherein each function $r_i : \mathbb{N}_0 \to \{0,1\}$ is such that $r_i(t) = 1$ if τ_i releases a job at time t, and $r_i(t) = 0$ otherwise.

[1] http://www.iasi.cnr.it/~vbonifaci/software.php

The release sequence R is said *legal* if the constraint on the job minimum interarrival times is met, that is:

$$\forall i = 1,\dots,n, \quad \forall t_r, t'_r \in \mathbb{N}_0,\ t_r < t'_r, \\ r_i(t_r) = 1 \wedge r_i(t'_r) = 1 \quad \Rightarrow \quad t'_r - t_r \geq P_i, \tag{2.1}$$

where t_r, t'_r represent two arbitrary release instants for τ_i in R.

Also, we define the *finishing sequence* F as a set of n functions

$$F = \{f_1(t),\dots,f_n(t) \mid t \in \mathbb{N}_0\},$$

wherein each function $f_i : \mathbb{N}_0 \to \{0,1\}$ is such that $f_i(t) = 1$ if a job of τ_i is completed at time t, and $f_i(t) = 0$ otherwise.

Set Q is defined by

$$Q = \{q_1(t),\dots,q_n(t) \mid t \in \mathbb{N}_0\},$$

wherein each function[2] $q_i : \mathbb{N}_0 \to \{0,1\}$ indicates if τ_i has a pending job at time t (in the run queue), defined by:

$$q_i(t) = \sum_{t'=0}^{t} r_i(t') - \sum_{t'=0}^{t} f_i(t'). \tag{2.2}$$

A schedule is represented by a set S of n functions

$$S = \{s_1(t),\dots,s_n(t) \mid t \in \mathbb{N}_0\},$$

with $s_i(t) = 1$ if any processor among the m available ones is allocated to τ_i over time $[t, t+1)$, and $s_i(t) = 0$ otherwise. With these notations, GFP schedule S is formally defined by

$$s_i(t) = 1 \quad \Leftrightarrow \quad q_i(t) > 0 \wedge \sum_{\ell=1}^{i-1} q_\ell(t) < m, \tag{2.3}$$

as well as the indicator function $f_i(t)$ of the finishing time for τ_i is defined by

$$f_i(t) = 1 \quad \Leftrightarrow \\ \exists t_r < t : \quad r_i(t_r) = 1 \wedge \sum_{t'=t_r}^{t-1} s_i(t') = C_i \wedge s_i(t-1) = 1 \tag{2.4}$$

Fig. 2.1 illustrates an example of a GFP schedule of $\mathscr{T} = \{\tau_1, \tau_2, \tau_3\}$. Below the schedule, we list the respective values for R, F, Q, and S, where the i-th row corresponds to task τ_i, and the j-th column corresponds to time instant $t = j-1$.

Time instants t_r, t_c are said to be the release and completion times of the same job of τ_i, if the

[2] We assume that no deadline miss has occurred by time t, meaning that τ_i has at most one pending job, due to $D_i \leq P_i$

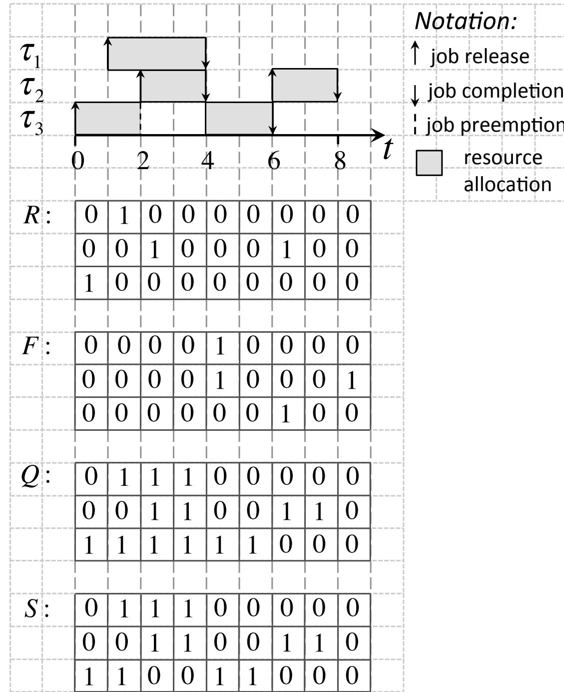


Figure 2.1: GFP schedule

following conditions hold:

$$r_i(t_r) = 1 \ \wedge \ f_i(t_c) = 1 \ \wedge \ \sum_{t=0}^{t_r} r_i(t) = \sum_{t=0}^{t_c} f_i(t), \tag{2.5}$$

meaning that i) some job of τ_i is released at time t_r, ii) some job of τ_i is completed at time t_c, and iii) the number of τ_i releases over time $[0, t_r]$ equals to the number of τ_i completions over time $[0, t_c]$.

We define the schedulability of $\mathscr{T}$ as follows.

Definition 1 (Schedulability of $\mathscr{T}$). *Let $\mathbb{L}_R$ denote all legal release sequences of task set $\mathscr{T}$, satisfying (2.1). $\mathscr{T}$ is said schedulable upon m processors, if for any $R \in \mathbb{L}_R$, all jobs of task τ_i, $i = 1, \ldots, n$, meet their deadlines:*

$$\forall R \in \mathbb{L}_R, \ \forall i \in \{1, \ldots, n\}, \ \forall (t_r, \, t_c), \qquad t_c - t_r \leq D_i, \tag{2.6}$$

where t_r, t_c are the respective release and completion times for the same job of τ_i, defined by (2.5).

2.4 Background on exact schedulability tests

Bonifaci and Marchetti-Spaccamela (2012) analyzed the schedulability of sporadic tasks by traversing a finite non-deterministic state transition graph, searching for a state with a violated deadline.

As our work aims at improving their approach, next we revisit the main ideas behind Bonifaci's work Bonifaci and Marchetti-Spaccamela (2012).

At a given time t, the state of the set of tasks is modeled by

$$(c_i, d_i, p_i)_{i=1}^{n} \in \mathbb{N}_0^{3n}, \tag{2.7}$$

where $c_i \in \{0,\ldots,C_i\}$ is the remaining execution time of τ_i pending job at t, if any; $d_i \in \{0,\ldots,D_i\}$ is the remaining time until its deadline; and $p_i \in \{0,\ldots,P_i\}$ is the remaining time until the earliest release of the next job of τ_i.

For tasks with implicit deadlines $D_i = P_i$, we can reduce the definition of a system state above by excluding parameter d_i:

$$(c_i, p_i)_{i=1}^{n} \in \mathbb{N}_0^{3n},$$

as $d_i = p_i$ always. Such an optimized representation of a state allows to reduce the amount of memory consumed by an algorithm for traversing a state transition graph.

A state transition graph for $\mathscr{T}$ is constructed as follows (see also Fig. 2.2). Each state in the graph represents a system state $(c_i, d_i, p_i)_{i=1}^{n}$ at a given time t. The initial state is $(0,0,0)_{i=1}^{n}$, meaning that no job has been yet released.

The state transition law, which governs the transition from state $g = (c_i, d_i, p_i)$ at time t to the next state $g' = (c'_i, d'_i, p'_i)$ at time $t+1$, is the following:

$$g' \in G' \iff \begin{cases} c'_i = c_i - s_i(t) + r_i(t+1)\,C_i \\ d'_i = \max(d_i - 1, 0) + r_i(t+1)\,D_i \\ p'_i = \max(p_i - 1, 0) + r_i(t+1)\,P_i, \end{cases} \tag{2.8}$$

where G' is a set of all successors for g at time $t+1$. In the equation above, $s_i(t)$ is the schedule function of τ_i uniquely determined by the system state g through (2.3), and $r_i(t+1)$ represents the release function (the "input" to the system) satisfying (2.1); that is

$$\begin{aligned} p_i - 1 > 0 \quad &\Rightarrow \quad r_i(t+1) = 0 \\ p_i - 1 = 0 \quad &\Rightarrow \quad r_i(t+1) \in \{0,1\}. \end{aligned} \tag{2.9}$$

State $(c'_i, d'_i, p'_i)_{i=1}^{n}$, at time t, is a scheduling failure state, if some job misses its deadline:

$$c'_i - r_i(t)\,C_i > d'_i - r_i(t)\,D_i, \tag{2.10}$$

with $r_i(t)$ defined by (2.9). This condition is true when the remaining execution time for a job (LHS of (2.10)) exceeds the remaining time until its deadline (RHS of (2.10)). $r_i(t)C_i$, $r_i(t)D_i$ are subtracted to correctly consider the case when the deadline of a job of τ_i coincides to the next release of τ_i.

Once the scheduling failure state is encountered, the algorithm reports unschedulability of $\mathscr{T}$, and terminates. If instead all feasible states have been checked, and no failure state has been

Algorithm 1 Exact schedulability test

```
1:  procedure EXACTSCHEDULABILITYTEST
2:      V ← ∅                                         ▷ initialize V
3:      G ← (0,0,0)_{i=1}^n                           ▷ initialize G
4:      while G ≠ ∅ do
5:          g ← Dequeue(G)
6:          compute G' for g                          ▷ from Eq. (2.8)
7:          for each g' ∈ G' do
8:              if g' ∉ V then
9:                  if ∃i, (2.10) holds then          ▷ deadline miss
10:                     return Unschedulable
11:                 end if
12:                 V = V ∪ {g'}
13:                 G = Enqueue(G, g')
14:             end if
15:         end for
16:     end while
17:     return Schedulable
18: end procedure
```

detected, then $\mathscr{T}$ is reported schedulable.

Algorithm 1 implements Bonifaci's test using breadth-first search (Cormen et al., 2009). It maintains two additional data structures: V is a set of checked states at previous iterations, and G is a FIFO queue, containing states for further examination.

Algorithm 1 works as follows. Set V of checked states is initially empty, and queue G contains only the initial state $g_0 = (0,0,0)_{i=1}^n$. At the first iteration of the while loop, the algorithm removes g_0 from G, and computes set G' of successors for g_0 using (2.8). Then, each state $g' \in G'$ is checked for a deadline miss (line 9), added to a list of checked states V (line 12), and added to queue G, to examine the g' successors at further iterations.

At each iteration of the while loop, the algorithm removes the first state g from queue G (line 5), and computes set G' of successors for g (line 6). Each state $g' \in G'$ that has not been checked yet (that is $g' \notin V$), is checked for a deadline miss, and added to V and G.

The algorithm terminates when queue G becomes empty, meaning that all feasible system states have been examined.

A higher runtime efficiency of Bonifaci's Algorithm 1 compared to other exact tests is mainly thanks to condition in line 8: each system state in a graph is checked only once. However, the same approach cannot be applied directly to tests using a timed automaton, due to specifics of a timed automaton. One solution has been proposed by Geeraerts et al. (2013), who derived a so called simulation relation technique for a timed automaton, but it does not seem to be more efficient than Bonifaci's approach.

Anyway, Algorithm 1 has exponential time and polynomial space complexity, and it might not terminate in a reasonable time even for small $\mathscr{T}$. According to our evaluation, Bonifaci's test is

Table 2.1: Bonifaci's test: Task set example

i	C_i	P_i	D_i
1	2	3	3
2	1	4	4
3	3	5	5

capable to deal with up to 5–6 tasks scheduled upon 2 processors, considering very small range of task periods, not exceeding 40.

Consider $\mathscr{T}$ with parameters reported in Table 2.1, to be scheduled by GFP upon $m = 2$ processors. In Fig. 2.2 we report the fragments of the state transition graph for such $\mathscr{T}$. The total number of distinguishable states in the full graph is 191.

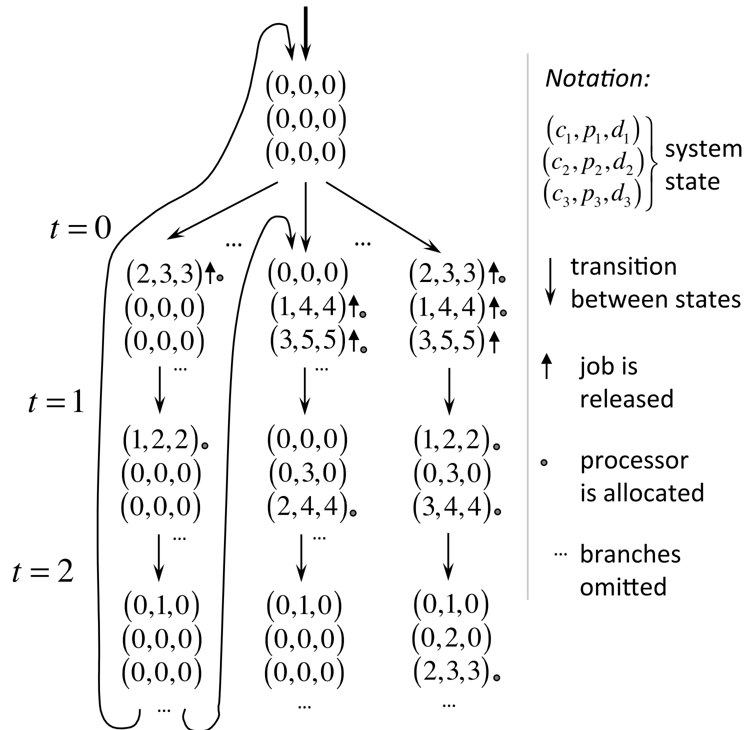


Figure 2.2: State transition graph

For the same $\mathscr{T}$, our test checks 12 states only, instead of 191, in one sixth of the running time relative to the test of Bonifaci and Marchetti-Spaccamela (2012), and the efficiency of our test increases for larger task sets.

2.5 An exact schedulability test using state space pruning

We next derive an improved exact schedulability test, exploring the idea of the state space pruning. The test checks schedulability of task τ_k, assuming that $\tau_1, \ldots, \tau_{k-1}$ are schedulable. Although we

consider GFP scheduler, most of the proposed improvements can be easily extended for other schedulers, such as GEDF.

2.5.1 Pruning constraints

Below we present a set of pruning methods, listed in the order of their decreasing complexity.

2.5.1.1 Job interference

We first show that, when analyzing schedulability of task τ_k, we can safely ignore any job of a higher-priority task τ_i, $i < k$, that causes no interference to any lower-priority tasks $\tau_{i+1}, \dots, \tau_k$.

Let us define job interference as follows.

Definition 2 (Job interference)**.** *Let J_i denote an arbitrary job of τ_i, with release and completion times denoted by t_r and t_c respectively. Job J_i is said to interfer with a lower priority job J_ℓ of τ_ℓ, $\ell > i$, if at some time $t \in [t_r, t_c)$ a processor is allocated to J_i, but not to J_ℓ:*

$$\begin{aligned} &\exists \ell > i, \quad \exists t \in [t_r, t_c): \\ &\qquad s_i(t) = 1 \,\wedge\, q_\ell(t) = 1 \,\wedge\, s_\ell(t) = 0, \end{aligned} \tag{2.11}$$

with $s_i(t)$, $q_\ell(t)$ defined by (2.3), (2.2).

We clarify this definition on an example. Let $\mathscr{T} = \{\tau_1, \dots, \tau_4\}$ be scheduled upon $m = 2$ processors. Consider the release sequence R for $\mathscr{T}$ as depicted in Fig. 2.3(a), and suppose that we analyze schedulability of task τ_4. Job $J_{1,1}$ of τ_1 interferes with job $J_{3,1}$ of τ_3 at time $t = 2$, as (2.11) holds:

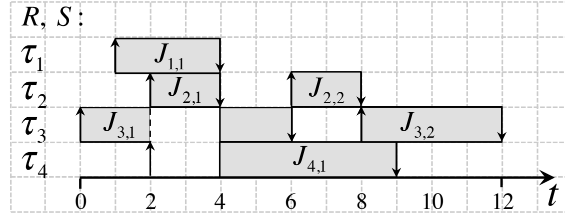
$$s_1(2) = 1 \,\wedge\, q_3(2) = 1 \,\wedge\, s_3(2) = 0.$$

Instead, job $J_{3,1}$ does not interfere with $J_{4,1}$, as (2.11) is violated. By removing such non-interfering jobs, we can produce a different arrival sequence that does not affect the schedule of task τ_4. Let us transform R, depicted in Fig. 2.3(a), into R', by erasing all jobs of task τ_i, $i < 4$, which do not interfere with lower priority jobs. These jobs are $J_{2,2}$, $J_{3,1}$, and $J_{3,2}$. Observe that the amount of resource available for τ_4 in R' is the same as in R.

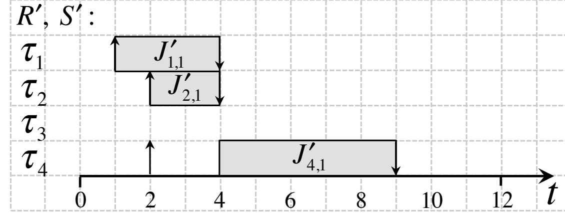
The next theorem generalizes such an observation.

Theorem 1. *Assume that $\tau_1, \dots, \tau_{k-1}$ are schedulable. Let $R = \{r_1(t), \dots, r_k(t)\}$ be any legal release sequence for $\mathscr{T}$, and let $J_{i,t}$ denote the τ_i job, released at time t. Let R' be a new release sequence that excludes all jobs of task τ_i from R, $i < k$, which violate the interference condition (2.11):*

$$\begin{aligned} R' = &\{r'_1(t), \dots, r'_k(t)\}: \\ & r'_i(t) = \begin{cases} 1, & \textit{if } r_i(t) = 1 \textit{ and } (2.11) \textit{ holds for } J_{i,t} \\ 0, & \textit{otherwise} \end{cases}, \\ & \qquad i = 1, \dots, k-1, \\ & r'_k(t) = r_k(t). \end{aligned} \tag{2.12}$$



(a) A legal release sequence R for $\mathscr{T}$



(b) Transformed released sequence R' for $\mathscr{T}$

Figure 2.3: Schedule transformation for Theorem 1 by excluding non-interfering jobs

Then, an arbitrary job of task τ_k misses its deadline in R iff it misses its deadline in R'.

The proof of Theorem 1 is provided in Appendix A.1. Although the proof is provided for GFP scheduler, it can be generalized for any other work-conserving scheduler, such as GEDF.

According to Theorem 1, the worst-case release sequence for τ_k is among those satisfying the following condition.

Corollary 1. *Let $\mathbb{R}^{\mathsf{reduced}}_{\mathscr{T}}$ denote all legal release sequences for $\mathscr{T}$, wherein each job of τ_i, $i = 1,\dots,k-1$, interfers with a lower priority job, as defined by (2.11). τ_k is schedulable for each legal R, satisfying (2.1), iff τ_k is schedulable for each $R' \in \mathbb{R}^{\mathsf{reduced}}_{\mathscr{T}}$.*

We next apply Corollary 1 to reduce the computation time of Algorithm 1. Let us extend the definition of a system state (2.7) at time t to

$$\left(c_i, d_i, p_i, b_i\right)_{i=1}^{k},$$

where b_i is boolean, such that $b_i(t) = 1$ iff τ_i has a pending job at t, and that job has interfered with a lower priority one by time t inclusive (meaning that condition (2.11) holds for that τ_i job at some time $t^* \leq t$).

The transition law (2.8) is extended for $b_i(t)$ accordingly. An initial state is $(0,0,0,0)_{i=1}^{k}$ with $b_i = 0$. For state $(c_i, d_i, p_i, b_i)_{i=1}^{k}$ at any time $t \geq 0$, the value of b_i is computed by

$$b_i = \begin{cases} 1, & \text{if } s_i(t) = 1 \land (\exists \ell < i : q_\ell(t) = 1 \land s_\ell(t) = 0) \\ 1, & \text{if } b_i^{\mathsf{prec}} = 1 \ \land \ q_i(t) = 1 \ \land \ r_i(t) = 0 \\ 0, & \text{otherwise} \end{cases}, \tag{2.13}$$

where b_i^{prec} corresponds to the preceeding state. In the definition above, $b_i = 1$ iff τ_i interfers with a lower priority job at time t (that is the first condition), or τ_i job, pending at time t, has interfered with a lower priority job prior to time t (that is the second condition).

We can determine if a job is non-interfering only after analyzing its entire execution. Therefore, we can check if (2.11) holds only when job completes in a state transition graph. Below we update the transition law (2.8) accordingly, by excluding from the analysis every state with jobs that violate (2.11).

Suppose that state $(c_i, d_i, p_i, b_i)_{i=1}^{k}$ at time t is such that

$$\exists \ell < k: \quad c_\ell = 1 \ \wedge \ s_\ell = 1 \ \wedge \ b_\ell = 0,$$

with s_ℓ computed by (2.3). Due to Corollary 1, we can safely discard a schedule with such a state from the analysis, because condition (2.11) is violated for τ_ℓ: $c_\ell > 0$ means that τ_ℓ has a pending job at time t, $b_\ell = 0$ means that that τ_ℓ job does not interfer with any lower priority job by time $t+1$ inclusive, and $c_\ell = 1 \wedge s_\ell = 1$ means that τ_ℓ job will be completed by time $t+1$.

Then, the transition law (2.8) for state g is optimized by adding a pruning constraint

$$\forall g' \in G', \ \forall i < k: \quad c'_i = 1 \ \wedge \ s'_i = 1 \ \longrightarrow \ b'_i = 1, \tag{2.14}$$

where $g' = (c'_i, d'_i, p'_i, b'_i)$ is a successor for g, with G' defined by (2.8).

We next provide an example, to illustrate the efficiency of constraint (2.14) in pruning the state space. Consider $\mathscr{T}$ with parameters reported in Table 2.1, scheduled upon $m = 2$ processors. Fig. 2.4 depicts a reduced state transition graph for such $\mathscr{T}$, thanks to (2.14). Another example, for the same $\mathscr{T}$, is depicted in Fig. 2.5(b), where we compare the performance of constraint (2.14) to other pruning constraints derived later in this work. For each pruning constraint, we specify the remaining number of states in a pruned graph. A more thorough evaluation of constraint (2.24) is provided later in Section 2.5.3.

Thanks to Theorem 1, we can bound the length of the longest release sequence for the schedulability analysis as follows.

Corollary 2. *When analyzing the schedulability of task τ_k, it is sufficient to test only the schedules of length not exceeding $\bar{t}$, with $\bar{t}$ defined by any of the following equations (the equations below assume that time is continuous):*

$$\bar{t} \ = \ \sum_{i=1}^{k} D_i \tag{2.15}$$

$$\bar{t} \ = \ \max\left(C_1, \ldots, C_m\right) \ + \ \sum_{i=m+1}^{k} D_i \tag{2.16}$$

$$\bar{t} \ = \ \max\left(C_1, \ldots, C_m\right) \ + \ \sum_{i=m+1}^{k-1} R_i \ + \ D_k, \tag{2.17}$$

where R_i denotes the worst-case response time for task τ_i, if known, otherwise we set $R_i = D_i$.

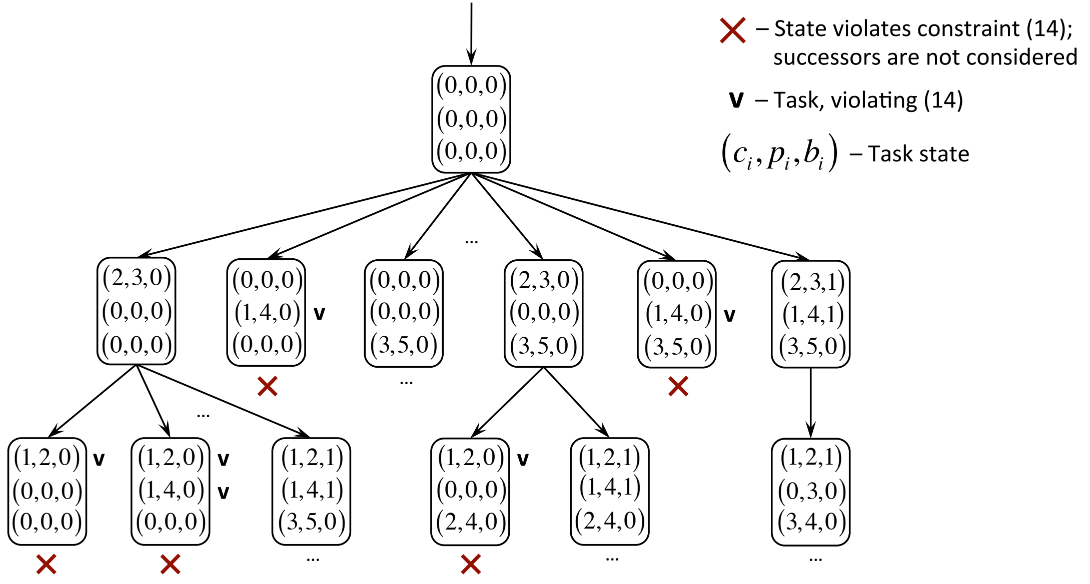


Figure 2.4: Pruned state transition graph by constraint (2.14) of interfering jobs.

If time is assumed discrete, then the equations for $\bar{t}$ above are reduced further to

$$\bar{t} = \sum_{i=1}^{k-1} (D_i - 1) + D_k$$

$$\bar{t} = \max(C_1, \ldots, C_m) - 1 + \sum_{i=m+1}^{k-1} (D_i - 1) + D_k \tag{2.18}$$

$$\bar{t} = \max(C_1, \ldots, C_m) - 1 + \sum_{i=m+1}^{k-1} (R_i - 1) + D_k, \tag{2.19}$$

Note that Corollary 2 applies to constrained-deadline tasks, scheduled by GFP scheduler only.

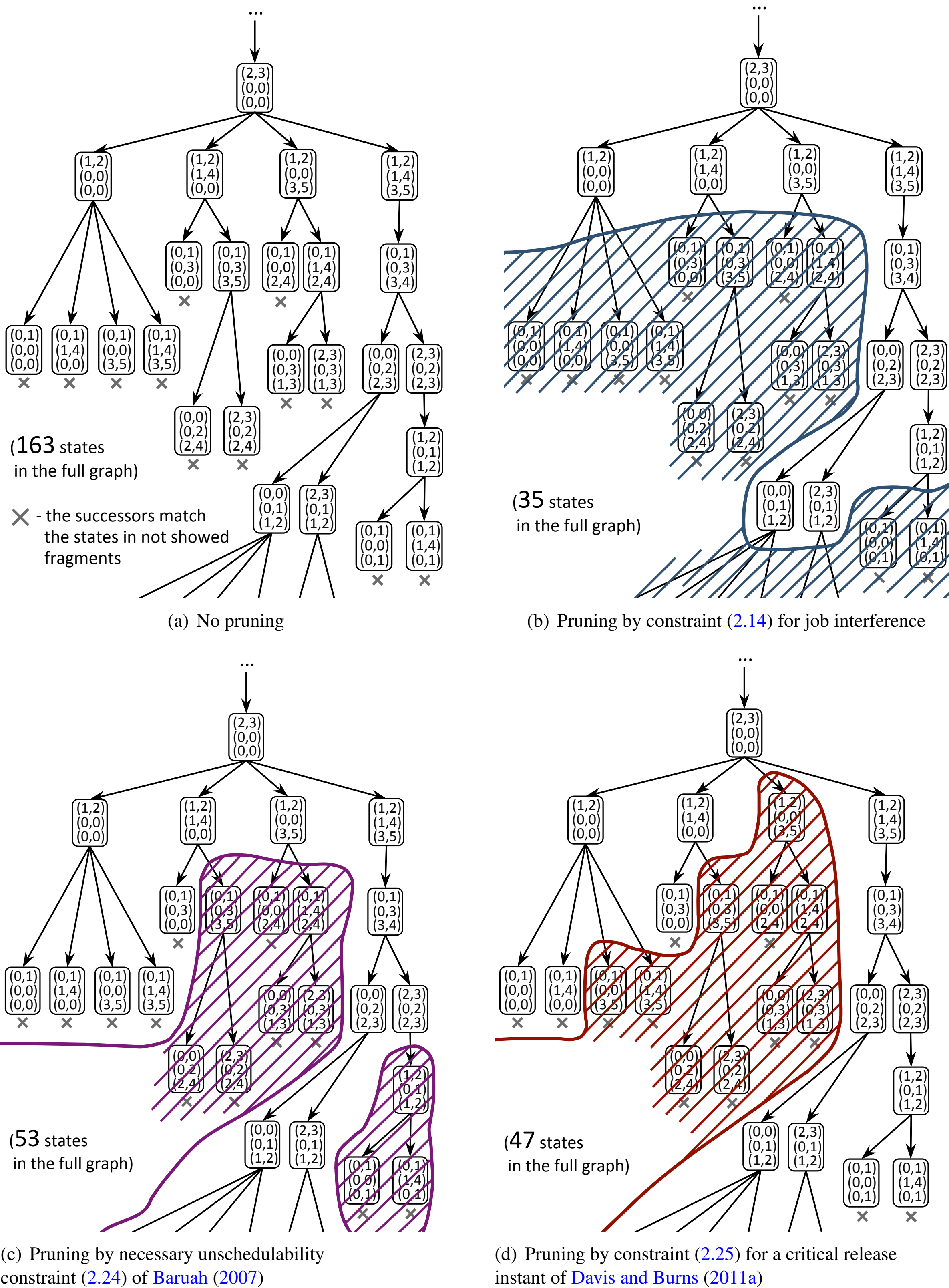
Bounds (2.15)-(2.16) originate the schedules depicted in Figs. (2.6(a)) and (2.6(b)). Observe that (2.15) does not depend on the number of available processors, and it applies to both the uniprocessor and multiprocessor cases. Instead, bounds (2.16) and (2.17) explore the number of available processors m, and they are tighter compared to (2.15).

The formal proof for Corollary 2 is provided in Appendix A.2.

2.5.1.2 Sufficient schedulability condition

Next, we prune the state space G' for Algorithm 1 through applying a sufficient schedulability condition. In fact, if a sufficient schedulability condition holds for some system state, then this state cannot lead to a failure state, and thus we do not need to examine its successors.

At time t, let state $(c_i, d_i, p_i)_{i=1}^{k}$ be such that $c_k > 0$, meaning that τ_k has a pending job at time t, with remaining execution time c_k and a deadline at time $t + d_k$.



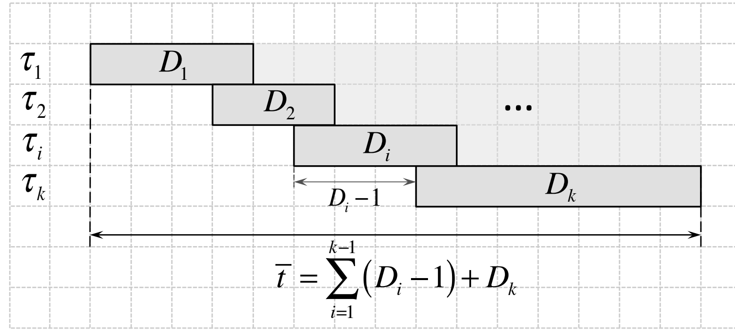
(a) No pruning

(b) Pruning by constraint (2.14) for job interference

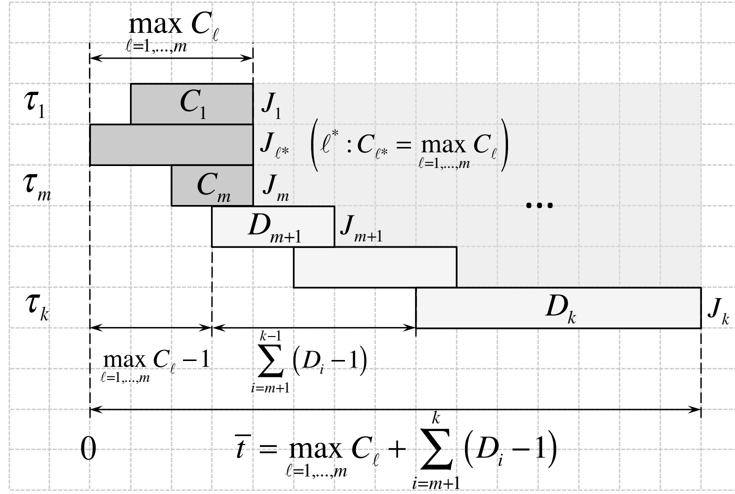
(c) Pruning by necessary unschedulability constraint (2.24) of Baruah (2007)

(d) Pruning by constraint (2.25) for a critical release instant of Davis and Burns (2011a)

Figure 2.5: The comparison of the performance of pruning constraints. Considered task parameters are reported in Table 2.1. The hatched area denotes the states pruned by the respective pruning constraint



(a) $\bar{t}$ not depending on the number of processors m



(b) Tighter $\bar{t}$ exploring the number of available processors m

Figure 2.6: An upper bound $\bar{t}$ for the length of the longest schedule, satisfying Corollary 1. The figures assume discrete time

Reusing the analysis of Baruah (2007), the amount of resource allocated to τ_k over time $[t, t + d_k)$ is at least

$$d_k - \bar{I}_k, \tag{2.20}$$

where $\bar{I}_k$ is the upper bound on interference, caused by $\tau_1, \ldots, \tau_{k-1}$ on τ_k, computed by (see Fig. 2.7)

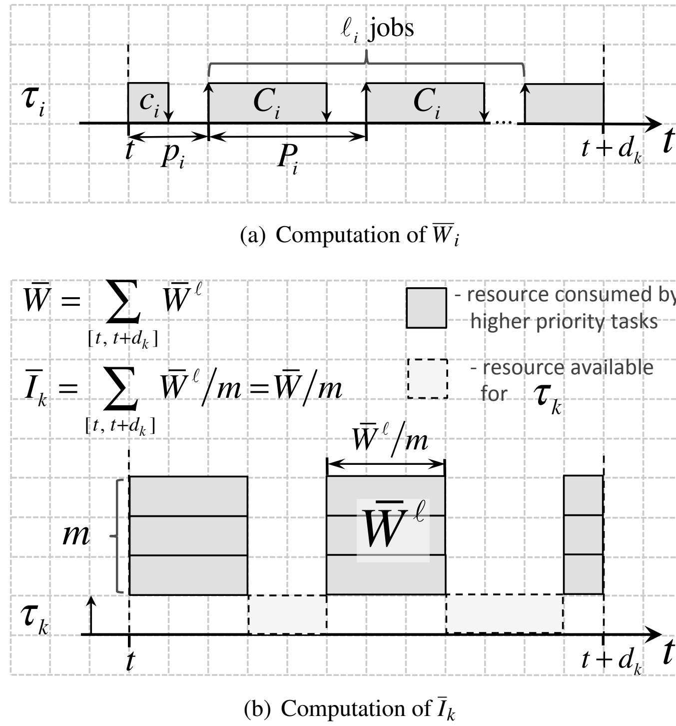
$$\bar{I}_k = \frac{\overline{W}}{m}, \tag{2.21}$$

with the maximum aggregated workload $\overline{W}$ for $\tau_1, \ldots, \tau_{k-1}$ computed by

$$\begin{aligned} \overline{W} &= \sum_{i=1}^{k-1} \overline{W}_i \\ \overline{W}_i &= \min(c_i, d_k) + \ell_i C_i + \min(C_i, \Delta_i), \end{aligned} \tag{2.22}$$

with

$$\ell_i = \max\left(0, \left\lfloor \frac{d_k - p_i}{P_i} \right\rfloor\right) \quad \text{and} \quad \Delta_i = d_k - p_i - \ell_i P_i. \tag{2.23}$$



(a) Computation of $\overline{W}_i$

(b) Computation of $\bar{I}_k$

Figure 2.7: Maximized job interference according to Baruah (2007)

If the lower bound (2.20) on supply allocated to τ_k is not lower than τ_k remaining demand c_k, that is $d_k - \bar{I}_k \geq c_k$, then τ_k is guaranteed schedulable. Thus, Algorithm 1 needs to examine the successor states for $(c_i, d_i, p_i)_{i=1}^k$, only if

$$d_k - \bar{I}_k < c_k. \tag{2.24}$$

Fig. 2.5(c) illustrates the performance of pruning constraint (2.24) for a task set $\mathscr{T}$ with parameters reported in Table 2.1. A more detailed evaluation of (2.24) is provided later in Section 2.5.3.

We have chosen condition (2.24) due to its low computation time. However, many other tests could be used instead of (2.24)[3], and their performance remains to be analysed. Another advantage of condition (2.24) is that it applies to any other scheduler, such as GEDF, after extending the summation in (2.22) for $i = 1, \ldots, n$, and $i \neq k$. Note however that a tighter pruning constraint of a higher computation cost will not necessarily outperform a less accurate pruning constraint of a lower computation cost.

2.5.1.3 Critical release instant

Davis and Burns (2011b) have shown that the worst-case execution scenario for a job of task τ_k occurs when that job is released at such time t ($r_k(t) = 1$), when all m processors are occupied by higher priority tasks $\tau_1, \ldots, \tau_{k-1}$ (that is $\sum_{\ell=1}^{k-1} q_\ell(t) \geq m$), but there is at least one processor idle

[3] Davis and Burns (2011a) provide a thorough survey of existing sufficient tests.

Table 2.2: Critical release instant: Task set example

i	C_i	P_i	D_i
1	3	6	6
2	4	6	6
3	2	3	3
4	-	12	12

during the preceeding time interval $[t-1,t)$ (that is $\sum_{\ell=1}^{k} q_\ell(t-1) < m$):

$$r_k(t) = 1 \quad \Rightarrow \quad \sum_{\ell=1}^{k-1} q_\ell(t) \geq m \;\wedge\; \sum_{\ell=1}^{k} q_\ell(t-1) < m, \tag{2.25}$$

with $q_\ell(t)$ defined by (2.2), and $q_\ell(t) = 0$ extended for $t < 0$. Further details can be found in Theorem 1 of Davis and Burns (2011b).

To illustrate the performance of pruning constraint (2.25), in Fig. 2.5(d) we depict a reduced state transition graph for a task set $\mathscr{T}$ with parameters reported in Table 2.1, thanks to (2.25).

We next adapt such an approach to restrict the release times for $\tau_1,\ldots,\tau_{k-1}$. We first provide an example. Consider $\mathscr{T} = \{\tau_1,\ldots,\tau_4\}$ with parameters reported in Table 2.2, scheduled upon $m = 2$ processors. We analyze schedulability of τ_4.

Fig. 2.8(a) depicts a legal release sequence for $\mathscr{T}$, denoted by R. In such R, job $J_{1,2}$ of τ_1 causes no interference to other jobs until time 10. Observe also that, after releasing $J_{1,2}$ at time 8, τ_1 cannot release another job until the deadline of a job of τ_4 at time 13. Then, without any optimistic assumption, we can safely postpone the release of $J_{1,2}$ until time 10, as depicted in Fig. 2.8(b).

The same reasoning does not apply, however, to job $J_{3,1}$ of τ_3: Delaying the release of $J_{3,1}$ might potentially affect the release time of consequtive $J_{3,2}$ (due to the constraint on the minimal time separation P_3), and $J_{3,2}$ in turn affects schedulability of τ_4.

We next formalize the discussion above. For an arbitrary release sequence R, let time t be such that:

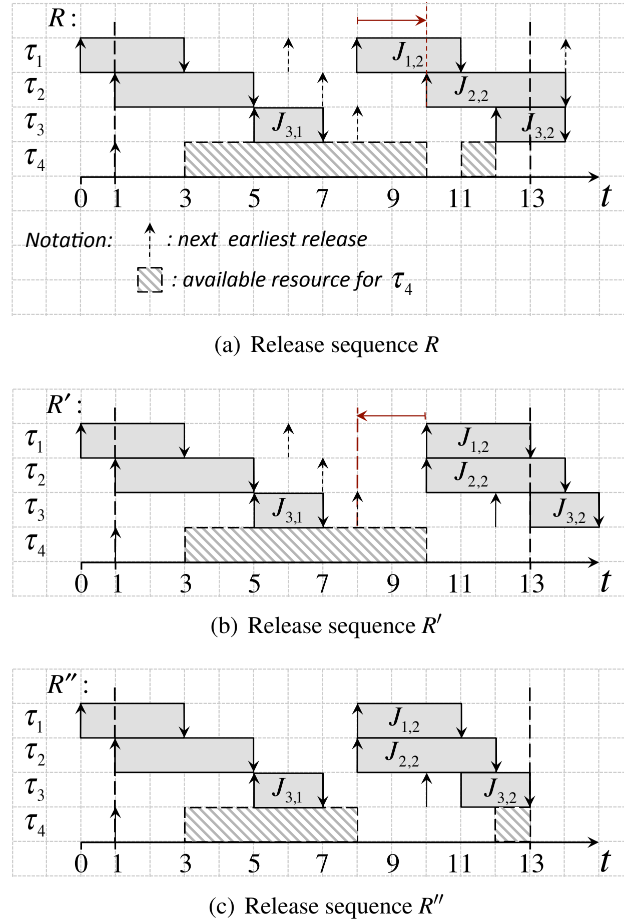
(a) τ_k has a pending job at time t: $q_k(t) = 1$;

(b) τ_i, with $i < k$, releases a job at time t: $r_i(t) = 1$;

(c) τ_i cannot release another job until τ_k's deadline:

$$P_i - D_k + (t - t_r) \geq 0, \tag{2.26}$$

where t_r denotes the release time for τ_k job, pending at time t.



(a) Release sequence R

(b) Release sequence R'

(c) Release sequence R''

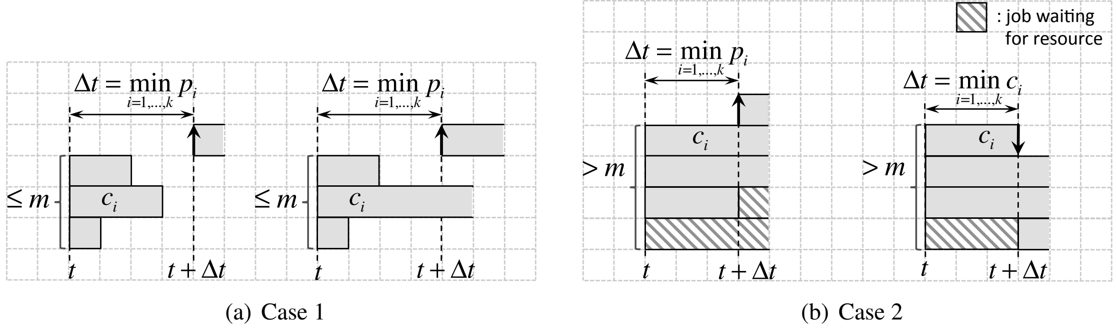
Figure 2.8: Critical release instants

At such t, there should be at least $m+1$ pending jobs for $\tau_1,\ldots,\tau_k$:

$$\forall t, i<k: \begin{cases} q_k(t)=1 \\ r_i(t)=1 \\ P_i - D_k + t - t_r \geq 0 \end{cases} \Rightarrow \sum_{\ell=1}^{k} r_\ell(t) > m. \tag{2.27}$$

The efficiency of (2.27) in pruning the state space is higher for lower ratios $^{P_k}/_{P_{\min}}$, with $P_{\min} = \min_{i=1,\ldots,k-1} P_i$; that is due to presence of (2.26) in (2.27).

The release times for $\tau_1,\ldots,\tau_{k-1}$ can be constrained further, by exploring their periods $P_1,\ldots,P_{k-1}$. Recall release sequence R', depicted in Fig. 2.8(b). At time 8, no job is released by $\tau_1,\ldots,\tau_3$, although each of them could; the next job is only released 2 time units later, at time 10. To maximize interference on τ_4, we can transform R' into a new R'', by shifting 2 time units left all release instants for $\tau_1,\ldots,\tau_3$, occurred after time 8, as depicted in Fig. 2.8(c). Clearly, interference on τ_k cannot decrease due to such a transformation.



(a) Case 1 (b) Case 2

Figure 2.9: Optimized clock transition Δt: cases

Thus, we require that each R satisfies the condition

$$\sum_{i=1}^{k-1} p_i(t) > 0, \qquad \forall t \tag{2.28}$$

in order to exclude case $\sum_{i=1}^{k-1} p_i(t) = 0$, when each $\tau_1, \ldots, \tau_{k-1}$ can release another job, but none of them does.

Note that, while conditions (2.25) and (2.27) for a critical release instant apply to GFP scheduler only, condition (2.28) applies to any other scheduler, such as GEDF, after extending the summation in (2.28) for $i = 1, \ldots, n$, and $i \neq k$.

We do not provide a formal proof for constraints (2.27) and (2.28). Such a proof can be conducted by analogy to the proof of Theorem 1 in Davis and Burns (2011b), which results in (2.25).

We conclude that set G' of successor states, defined by transition law (2.8), can be pruned further, by adding constraints (2.25), (2.27), (2.28).

A detailed evaluation of pruning constraints (2.25), (2.27), (2.28) is provided later in Section 2.5.3.

2.5.1.4 Optimized clock transition

To avoid unnecessary check for a deadline miss at every time instant, we next propose an optimized clock transition between checked system states.

Let $g = (c_i, d_i, p_i)_{i=1}^{k}$ denote an arbitrary system state. Suppose that condition (2.10) for a failure state does not hold for g, that is

$$c_i \quad \leq \quad d_i, \qquad i = 1, \ldots, k. \tag{2.29}$$

We next analyze the following cases for g: either at most m jobs are pending, or more than m jobs are pending, where m is the number of available processors.

Suppose first that at most m jobs are pending for state g, that is the following condition holds:

$$\sum_{i=1}^{k} \min(c_i,\ 1) \quad \leq \quad m,$$

with $c_i \geq 1$ if τ_i has a pending job, and $c_i = 0$ otherwise.

For such g, as the number of pending jobs does not exceed the number of processors, and condition (2.29) holds, then no deadline miss will occur, until another job is released (see also Fig. 2.9(a)). The next earliest job release will occur only Δt time units later, with Δt defined by

$$\Delta t = \left(\min_{i=1,\ldots,k} p_i \right)_1, \tag{2.30}$$

where $(x)_1$ denotes $\max(1,x)$. Thus, for state g with at most m pending jobs, the next state to be checked is Δt time units later.

Suppose instead that more than m jobs are pending at state g. Let Δt denote the remaining time until the next system event occurs, such as job release, completion, or deadline (see also Fig. 2.9(b)):

$$\Delta t \quad = \quad \left(\min_{i=1,\ldots,k} (c_i > 0\ ?\ c_i\ :\ p_i) \right)_1, \tag{2.31}$$

with a conditional operator $(c_i > 0\,?\,c_i\ :\ p_i)$ returning c_i, if $c_i > 0$, and p_i otherwise.

Let $g' = (c'_i, d'_i, p'_i)_{i=1}^{k}$ denote any successor state for state g, which is Δt time units later, with Δt defined as above. If any of intermediate states between g and g' is a failure state, then g' is a failure state as well. Thus, we can discard all intermediate states between states g and g' from the analysis, without making any optimistic assumption.

We conclude that an optimized clock transition Δt between system states is computed by (2.30) and (2.31):

$$\Delta t = \begin{cases} \left(\min_{i=1,\ldots,k} p_i \right)_1, & \text{if } \sum_{i=1}^{k} \min(c_i, 1) \leq m \\ \left(\min_{i=1,\ldots,k} (c_i > 0\ ?\ c_i\ :\ p_i) \right)_1, & \text{otherwise.} \end{cases} \tag{2.32}$$

The transition law (2.8) is updated accordingly, by replacing clock increment "1" in (2.8), (2.9) by Δt:

$$(c'_i, d'_i, p'_i)_{i=1}^{k} \in G' \iff$$

$$\begin{cases} c'_i = (c_i - s_i(t)\,\Delta t)_0 + r_i(t+\Delta t)\,C_i \\ d'_i = (d_i - \Delta t)_0 + r_i(t+\Delta t)\,D_i & \quad i = 1,\ldots,k \\ p'_i = (p_i - \Delta t)_0 + r_i(t+\Delta t)\,P_i \end{cases} \tag{2.33}$$

where $g' = (c'_i, d'_i, p'_i)$. Δt and $s_i(t)$ are defined by (2.32), (2.3), and $r_i(t+\Delta t)$ satisfies (2.1):

$$\begin{aligned} p_i - \Delta t > 0 \quad &\Rightarrow \quad r_i(t+\Delta t) = 0 \\ p_i - \Delta t = 0 \quad &\Rightarrow \quad r_i(t+\Delta t) \in \{0,1\} \end{aligned}$$

2.5.2 Test procedure

The optimized procedure for an exact schedulability test is as follows. In Algorithm 1, we replace the state transition law (2.8) by (2.33), as well as incorporate pruning constraints (2.14), (2.24), (2.25), (2.27), (2.28) into (2.33).

A thorough evaluation of the improved test is provided in Section 2.5.3, confirming its significantly lower runtime and memory consumption, compared to the test of Bonifaci and Marchetti-Spaccamela (2012).

2.5.3 Evaluation

We finally evaluate the performance of the exact schedulability test, presented in Section 2.5.2. The test is implemented using C++ libraries, by extending Bonifaci's tool. The implementation of our test is publicly available[4].

The experiments are conducted on a hardware platform with the following specifications:

- Processor: Intel Core i7-4710MQ CPU @ 2.5GHz

- Operating memory (RAM): 15,50 GB 1600 MHz

- System type: 64-bit

- Operating system: Ubuntu 14.04 LTS

2.5.3.1 Task set generation

Sporadic task sets $\mathscr{T} = \{\tau_i = (C_i, P_i)\}$ with implicit deadlines $D_i = P_i$ are randomly generated by specifying the number of tasks n, the total task set utilization $U_{\mathscr{T}}$, the maximum individual task utilization $U_{\max}$, and the range for task periods $[P_{\min}, P_{\max}]$.

The minimum period $P_{\min}$ is randomly taken from range $[3;10]$, and all task periods are generated such that the specified ratio $P_{\max}/P_{\min}$ holds. Task execution times C_i are chosen by solving

[4] www.cister.isep.ipp.pt/docs/CISTER-TR-150503

Table 2.3: Key parameters: default values

Settings		
Number of processors, m	2	3
Number of tasks, n	5	7
Task set utilisation, $U_{\mathcal{T}}$	1.6	2.2
Maximum individual task utilisation, $U_{\max}$	0.6	0.6
Minimum task period, $P_{\min}$	[3, 10]	[3, 10]
Ratio between the maximum and minimum task periods, $P_{\max}/P_{\min}$	4	4

the following linear integer optimization problem, using CPLEX:

$$\begin{aligned}
\text{minimize} \quad & \left| U_{\mathcal{T}} - \sum_{i=1}^{n} C_i/P_i \right| + |U_{\max} - C_{i^*}/P_{i^*}| \\
\text{subject to} \quad & \\
& 0 < C_i < P_i, \qquad i = 1, \ldots, n \\
& \left| U_{\mathcal{T}} - \sum_{i=1}^{n} C_i/P_i \right| \leq \delta_{U_{\mathcal{T}}} U_{\mathcal{T}} \\
& |U_{\max} - C_{i^*}/P_{i^*}| \leq \delta_{U_{\max}} U_{\max} \\
& C_i/P_i \leq C_{i^*}/P_{i^*}, \qquad i = 1, \ldots, n,
\end{aligned}$$

where C_i, $i = 1, \ldots, n$, are integer optimization variables, $|x|$ denotes the absolute value of x, and index i^* corresponds to task τ_{i^*} having the maximum utilization $U_{\max}$; i^* is randomly taken from range $[1, \ldots, n]$.

We allow a relative deviation $\delta_{U_{\mathcal{T}}} = 1.5\%$ between the specified value U and the actual tasks utilization $\sum_{i=1}^{n} C_i/P_i$, as well as deviation $\delta_{U_{\max}} = 2.5\%$ between the specified value $U_{\max}$ and the actual maximum task utilization $\max_{i=1,\ldots,n} C_i/P_i$.

We randomly generate task sets for parameters reported in Table 3.3. In each experiment, one key parameter varies, while the rest are left equal to the default values in Table 3.3.

The choice for parameter values is constrained by a hardware limitation of 16 Gb of operating memory. $U_{\mathcal{T}}$ is chosen such that the pessimism of the sufficient test by Guan et al. (2009) is maximized (that is the case when usage of an exact test makes more sense). For a thorough evaluation, the pessimism of Guan's test is analyzed for varying $U_{\mathcal{T}}$ as well.

2.5.3.2 Experiments: Runtime reduction

We first compare the runtime of our test against Bonifaci's test[5], for $m = 2$. Figs. 2.10(a), 2.10(b) report an average runtime for a varying number of tasks n, and for a varying ratio $p_{\mathsf{max}}/p_{\mathsf{min}}$ of task periods; the depicted results consider only those task sets, determined as schedulable. We confirm a significantly lower computation time of our test, which allows to analyze task sets with larger n and $p_{\mathsf{max}}/p_{\mathsf{min}}$. However, runtime complexity remains exponential in n.

The second experiment is conducted for $m = 3$. The average runtime for our test is reported in Figs. 2.10(c), 2.10(d). For such settings, Bonifaci's test requires more than 16 Gb of memory in most cases, so that the comparison to our test is infeasible. In 5% of cases, our test exceeds 16 Gb as well, and those cases are discarded.

To analyze contribution of each of pruning constraints (2.12), (2.24)-(2.33) into the runtime reduction of our test, we have conducted a series of additional experiments. The results are reported in Figs. 2.12(b)-2.12(d), for $m = 2$. The plotted size reduction is computed by

$$\mathsf{reduction}_{(x)} = \frac{N_{(x)\ \mathrm{excluded}}}{N},$$

where N is the number of states checked by the algorithm employing all pruning constraints (2.14), (2.24), (2.25)-(2.28), (2.33), and $N_{(x)\ \mathrm{excluded}}$ is the number of states checked by the same algorithm excluding the pruning constraint (x).

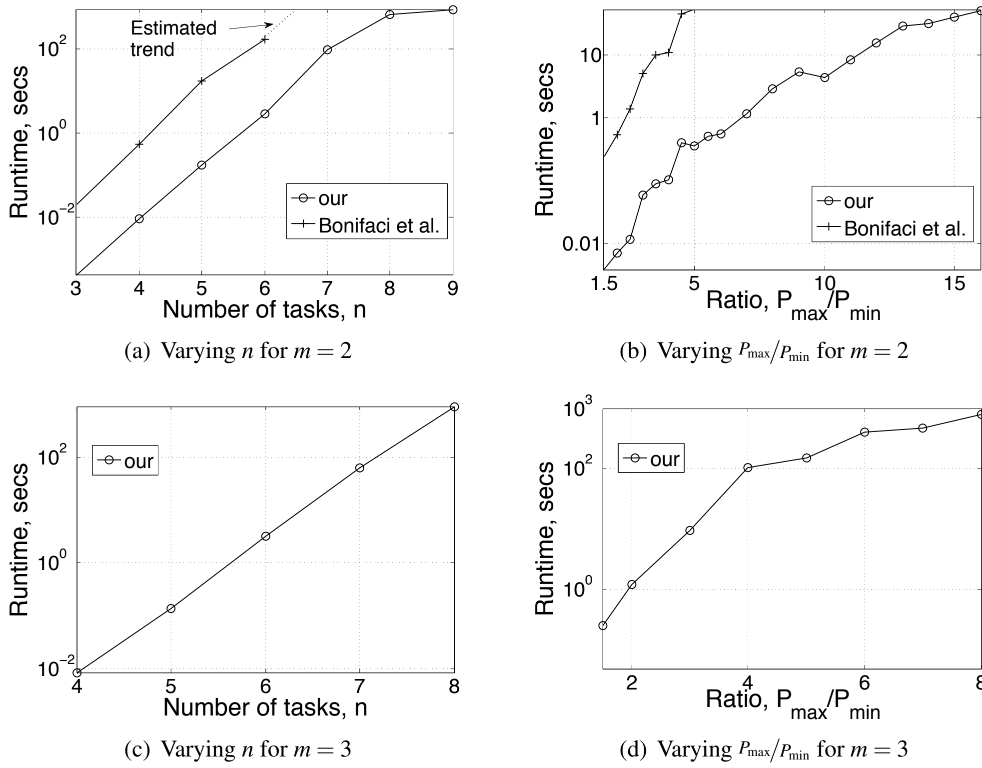
Despite of the polynomial space complexity, our evaluation shows that the required system memory for our algorithm (as for all other existing exact tests) increases significantly with the number of tasks and their periods. For example, for task sets comprised of just 10 tasks, and their periods not exceeding 40, the required memory already exceeds 16 GB in most cases.

2.5.3.3 Experiments: Comparison of exact and sufficient tests

To motivate the need for an exact test, we also evaluate the pessimism of Guan's sufficient test Guan et al. (2009), which in turn outperforms most of other existing sufficient tests[6]. The experiments are conducted when varying n and $U_{\mathscr{T}}$, for both $m = 2$ and $m = 3$. The results are reported in Figs. 2.11(a)-2.11(d). We confirm a significant pessimism of Guan's test, exceeding 50% under certain settings. However, our evaluation is limited to very small task sets, due to the high memory consumption of the exact test.

[5] We speeded-up the original code available at `http://www.iasi.cnr.it/~vbonifaci/software.php` by a factor 10–20 times, by recompiling it at optimized settings.

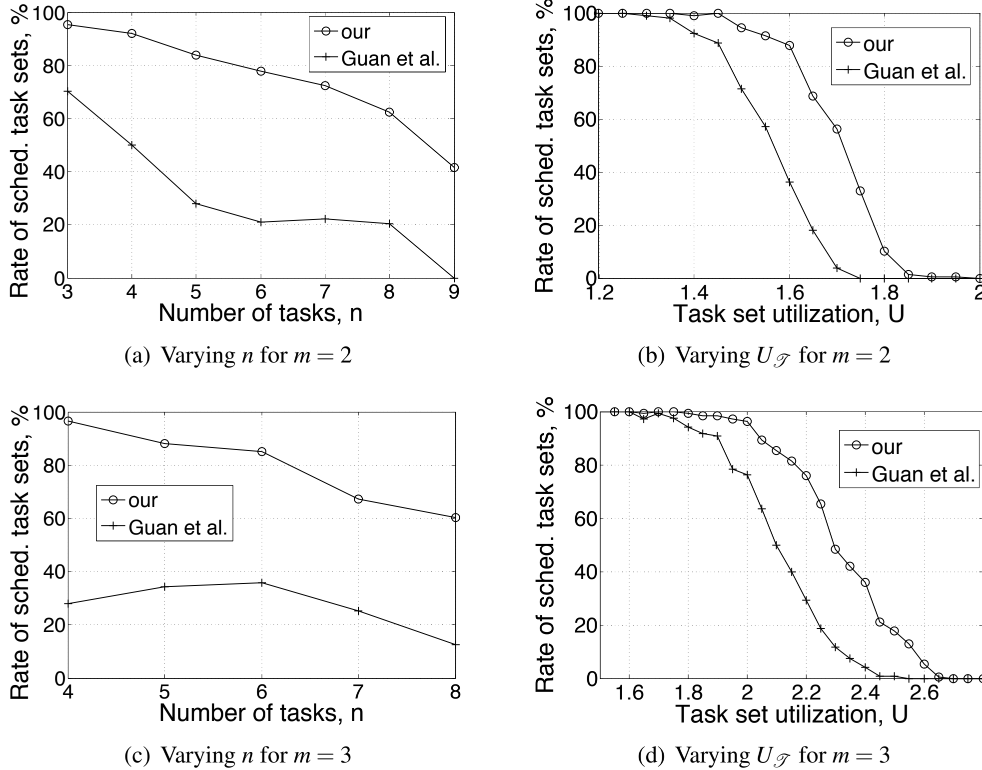
[6] We have implemented Guan's test following an optimized procedure in Davis and Burns (2011a)



(a) Varying n for $m = 2$ (b) Varying $P_{\max}/P_{\min}$ for $m = 2$

(c) Varying n for $m = 3$ (d) Varying $P_{\max}/P_{\min}$ for $m = 3$

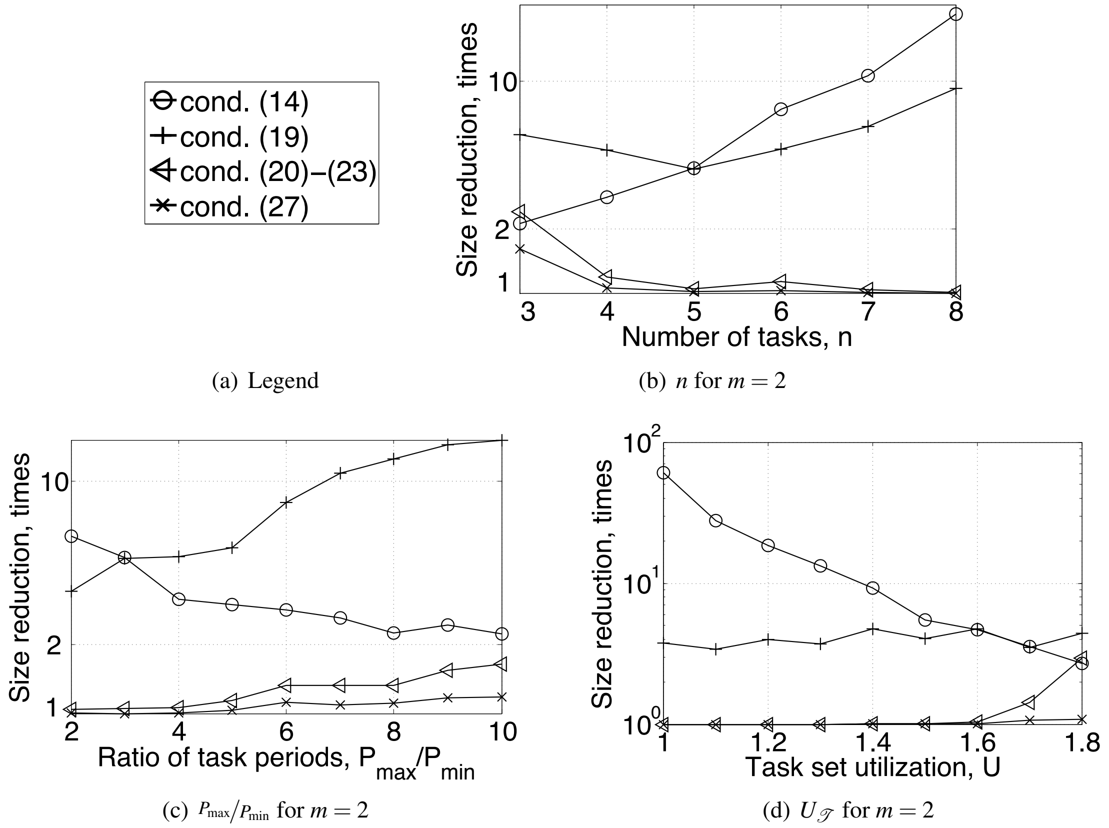
Figure 2.10: Evaluation: runtime of exact tests



(a) Varying n for $m = 2$ (b) Varying $U_{\mathcal{T}}$ for $m = 2$

(c) Varying n for $m = 3$ (d) Varying $U_{\mathcal{T}}$ for $m = 3$

Figure 2.11: Evaluation: performance of Guan's sufficient test



(a) Legend

(b) n for $m = 2$

(c) $P_{\max}/P_{\min}$ for $m = 2$

(d) $U_{\mathscr{T}}$ for $m = 2$

Figure 2.12: Evaluation: comparison of performance of pruning constraints (logarithmic scale)

2.6 An exact schedulability test for continuous-time schedulers using linear programming (LP)

The fundamental assumption behind the exact schedulability test proposed in Section 2.5.2 is that scheduling decisions can only be taken at discrete time instants $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$. However, the assumption of discrete time, rather than continuous, significantly complicates the problem, as no efficient methods, except enumeration-based ones, exist for dealing with discrete parameters. Aiming at deriving a faster test, we next relax the assumption of discrete time, allowing scheduling decisions to be taken anytime, and then employ linear programming (LP) methods to our test.

2.6.1 Schedule model

Let us first determine the length of the longest release sequence to be examined by the test, for continuous-time scheduling. For this, the bound (2.18) is extended by assuming no time granularity as follows:

$$\bar{t} = \max_{\ell=1,\ldots,m} C_\ell + \sum_{i=m+1}^{n} D_i. \tag{2.34}$$

Let $t_1,\ldots,t_p$ denote time instants in $[0,\bar{t}]$, when scheduling events occur, that is some job is released or completed. Assume that p is fixed, and $t_1,\ldots,t_p$ are continuous variables, such that

$$t_j < t_{j+1}, \qquad t_{p-1} < \bar{t} \le t_p \tag{2.35}$$

Next, the definitions of sequences R, F, S, originally defined in Section 2.3, are adjusted as follows. Let $R = [r_{i,j}]$ be the matrix of releases, with $i = 1,\ldots,n$ and $j = 1,\ldots,p$, such that $r_{i,j} = 1$ if τ_i releases a job at time t_j, and $r_{i,j} = 0$ otherwise. Also, the matrix of finishing times $F = [f_{i,j}]$ is such that $f_{i,j} = 1$ if any τ_i job is completed at time t_j, and $f_{i,j} = 0$ otherwise. Finally, $S = [s_{i,j}]$ is such that $s_{i,j} = 1$ if τ_i is scheduled over time $[t_j,t_{j+1})$, and $s_{i,j} = 0$ otherwise.

Valid matrices (R,F,S) must satisfy the following constraints. First, some job is released or completed, at each t_j:

$$\sum_{\ell=1}^{n} r_{i,j} + \sum_{\ell=1}^{n} f_{i,j} \ge 1, \qquad i = 1,\ldots,k, \quad j = 1,\ldots,p. \tag{2.36}$$

Also, release instants of τ_i must be separated by at least P_i time units. Let $t_{\ell(i,j)}$ denote the latest release instant of τ_i, preceeding time instant t_j (that is $t_{\ell(i,j)} < t_j$), so that index $\ell(i,j)$ is defined by

$$\ell(i,j) = \max\{x : x < j,\ r_{i,x} = 1\}. \tag{2.37}$$

Then, the constraint for consecutive releases of τ_i is

$$r_{i,j} = 1 \ \wedge \ \sum_{j'=1}^{j-1} r_{i,j'} > 0 \quad \Rightarrow \quad t_j - t_{\ell(i,j)} \ge P_i, \qquad i = 1,\ldots,k, \quad j = 1,\ldots,p, \tag{2.38}$$

where condition $\sum_{j'=1}^{j-1} r_{i,j'} > 0$ holds if τ_i has released a job prior to time t_j exclusive.

GFP scheduler allocates a processor to a job of τ_i when and only when the number of higher priority jobs, that is of tasks $\tau_1,\ldots,\tau_{i-1}$, is less than the number of processors m:

$$s_{i,j} = 1 \quad \Leftrightarrow \quad q_{i,j} = 1 \wedge \sum_{\ell=1}^{i-1} q_{\ell,j} < m, \qquad i = 1,\ldots,k, \quad j = 1,\ldots,p, \tag{2.39}$$

where $q_{i,j} \in \{0,1\}$ indicates if τ_i has a pending job at time t_j [7], and is defined by:

$$q_{i,j} = \sum_{\ell=1}^{j} r_{i,\ell} - \sum_{\ell=1}^{j} f_{i,\ell}.$$

A job of τ_i completes execution by time it receives C_i resource units. Considering that the aggregated amount of resource allocated to task τ_i over an arbitrary time interval $[t_{\ell(i,j)},t_j)$ is

[7] We assume that τ_i has at most one pending job over time $[t_{\ell(i,j)},t_j)$, due to $D_i \le P_i$ and no deadline miss occurred by time t_{j-1} inclusive.

computed by

$$\sum_{\ell=\ell(i,j)}^{j-1} s_{i,\ell}\,(t_{\ell+1}-t_\ell),$$

the constraint for τ_i job completion is as follows:

$$\begin{aligned} f_{i,j}=1 \quad &\Rightarrow \quad \sum_{\ell=\ell(i,j)}^{j-1} s_{i,\ell}\,(t_{\ell+1}-t_\ell)=C_i \quad \wedge \quad s_{i,j-1}=1, \\ f_{i,j}=0 \quad &\Rightarrow \quad \sum_{\ell=\ell(i,j)}^{j-1} s_{i,\ell}\,(t_{\ell+1}-t_\ell)<C_i \quad \vee q_{i,j-1}=0 \end{aligned} \tag{2.40}$$

with functions $q_{i,j-1}$ and $\ell(i,j)$ defined as above.

Observe that all constraints (2.35)–(2.40) are linear in variables $t_1,\ldots,t_p$.

With a given number p of time instants $t_1,\ldots,t_p$, when scheduling decisions are taken, and given matrices (R,F,S), let $\mathbb{T}_p$ denote the set of all cases $(t_1,\ldots,t_p)$, satisfying constraints (2.35)–(2.40).[8] We say that matrices (R,F,S) are feasible if and only if

$$\mathbb{T}_p \neq \emptyset. \tag{2.41}$$

The constraint above can be checked by any LP solver, typically in polynomial time.

2.6.2 Procedure of the test

We are now ready to adapt Bonifaci's Algorithm 1 to continuous-time scheduling, using the notation described above.

Recall the state transition graph introduced in Section 2.4. Let us adjust the definition (2.7) of a system state g at time t_j to

$$g=\{(c_i,d_i,p_i)_{i=1}^k,\,\mathbb{T}_j\} \tag{2.42}$$

with set $\mathbb{T}_j\subset\mathbb{R}^j$ of cases $(t_1,\ldots,t_j)$ defined as above, and parameters c_i, d_i, p_i expressed as linear functions in variables $t_1,\ldots,t_j$:

$$c_i=\left(\sum_{\ell=1}^{j} r_{i,\ell}\right)C_i-\sum_{\ell=1}^{j-1} s_{i,\ell}\,(t_{\ell+1}-t_\ell) \tag{2.43}$$

$$d_i=\left(t_{\ell(i,j)}+D_i-t_j\right)_0 \tag{2.44}$$

$$p_i=\left(t_{\ell(i,j)}+P_i-t_j\right)_0, \tag{2.45}$$

with $t_{\ell(i,j)}$ defined by (2.37).

It follows that state g is a failure state for τ_k, if

$$\mathbb{T}_j\bigcap\{c_k>d_k\}\neq\emptyset, \tag{2.46}$$

[8] In fact, region $\mathbb{T}_p$ is a p-dimentional polytope, defined as an intersection of linear constraints (2.35)–(2.40) in variables $t_1,\ldots,t_p$.

meaning that exists such a case $(t_1,\ldots,t_j) \in \mathbb{T}_j$, that some job of τ_k misses its deadline The constraint above can be checked by any LP solver, by solving a linear feasibility problem in variables $t_1,\ldots,t_j$.

Algorithm 1, for discrete-time scheduling, assumes that values of parameters c_i, d_i, p_i are always known, so that it is possible to check if two states $(c_i,d_i,p_i)_{i=1}^k$ and $(c'_i,d'_i,p'_i)_{i=1}^k$ are equal. We instead have redefined c_i, d_i, p_i as linear functions (2.43)-(2.45) in continuous variables $(t_1,\ldots,t_j) \in \mathbb{T}_j$, so that parameters c_i, d_i, p_i are not fixed to any specific values, but rather defined by a j-dimentional polytope in $t_1,\ldots,t_j$, which is bounded by constraints (2.43)-(2.45). Thus, we next need to decide on how to compare two graph states using such a notation.

Let $\mathbb{G} = \{g_\ell\}$ denote an arbitrary set of states, with each state denoted by $g_\ell = \{(c_i^\ell,d_i^\ell,p_i^\ell)_{i=1}^k, \mathbb{T}_{j_\ell}^\ell\}$. We say that state $g = \{(c_i,d_i,p_i)_{i=1}^k, \mathbb{T}_j\}$ is contained in set $\mathbb{G}$ (what is denoted by $\{g\} \subseteq \mathbb{G}$), if for each case $(t_1,\ldots,t_j) \in \mathbb{T}_j$, there exists such a state $g_\ell \in \mathbb{G}$, that has parameters c_i^ℓ, d_i^ℓ, p_i^ℓ equal to c_i, d_i, p_i respectively, for each $i = 1,\ldots,k$:

$$\{g\} \subseteq \mathbb{G} \iff \begin{array}{l} \forall (t_1,\ldots,t_j) \in \mathbb{T}_j \\ \quad \exists g_\ell \in \mathbb{G}, \quad \exists (t_1^\ell,\ldots,t_{j_\ell}^\ell) \in \mathbb{T}_{j_\ell}^\ell : \\ \quad\quad c_i = c_i^\ell \wedge d_i = d_i^\ell \wedge p_i = p_i^\ell, \qquad i = 1,\ldots,k \end{array} \tag{2.47}$$

We do not yet have an efficient mechanism for checking the constraint above; we keep it as a future work. Note that the constraint above is essential for deriving a fast schedulability test, as it allows to reduce the space complexity of the test from exponential to just polynomial. More details on space complexity are provided in Section 2.4.

To reduce the complexity of the test further, we also employ pruning constraints (2.12), (2.24)--(2.28): originally derived for discrete-time, these constraints directly apply to continuous-time scheduling as well.

The resulted procedure for schedulability test is reported in Algorithm 2. It recursively checks for all values $p \geq 1$, until condition (2.35) holds (line 13), and determines the existance of such a case (R,F,S), satisfying (2.41), (2.12), (2.24)-(2.28) (lines 13, 14), that condition (2.46) for a deadline miss holds (line 15). If such a case is found, then the test terminates immediately, reporting unschedulability of τ_k. If instead (2.46) never holds, then τ_k is reported schedulable.

Although not yet been implemented, such a test is expected to have a better scalability, compared to the test described in Section 2.5.2, as it only checks release scenarios for $\mathscr{T}$ with distinguishable orders of job arrivals, rather than checking all possible legal release scenarios.

2.7 An exact schedulability test using constraint programming (CP)

Another way to analyze schedulability of $\mathscr{T}$, instead of solving the LP problems formulated in Section 2.6, is to use constraint programming (CP) methods (Marriott and Stuckey, 1998). The major difference of constraint programming from mathematical programming is the support of

Algorithm 2 Schedulability test for τ_k

```
 1: procedure EXACTSCHEDULABILITYTEST
 2:     R,F,S ← [∅]                                   ▷ initialize to empty matrices
 3:     compute t̄                                      ▷ from Eq. (2.34)
 4:     RECURSE(1,R,F,S)
 5:     terminate τ_k is schedulable
 6: end procedure

 7: procedure RECURSE(p,R,F,S)
 8:     for ∀i ≤ k, ∀(r_ip, f_ip, s_ip) : r_ip, f_ip, s_ip ∈ {0,1} do
 9:         R = appendColumn(R, {r_ip}_{i=1}^k)
10:         F = appendColumn(F, {f_ip}_{i=1}^k)
11:         S = appendColumn(S, {s_ip}_{i=1}^k)
12:         compute T_p                                ▷ from Eq. (2.36)-(2.40)
13:         if T_p ∩ {t_{p-1} < t̄ ≤ t_p} ≠ ∅ then       ▷ due to (2.35)
14:             if (2.12), (2.24)-(2.28) hold then      ▷ pruning
15:                 if (2.46) holds then                ▷ deadline miss
16:                     terminate τ_k is unschedulable
17:                 end if
18:                 RECURSE(p+1,R,F,S)
19:             end if
20:         end if
21:     end for
22: end procedure
```

logical constraints, along with traditional arithmetic constraints. Below we will show that usage of logical constraints significantly simplifies the modeling of schedulability analysis problems.

Constraint programming is an emerging domain of operations research, and its efficiency has been already confirmed on a range of scheduling problems, such as those in process manufacturing. A wide range of CP solvers is available, e.g. GECODE and IBM CP Optimizer.

Below we briefly formulate an exact schedulability analysis problem using constraint programming.

First we define boolean function $\mathsf{equals}(x,y)$ as

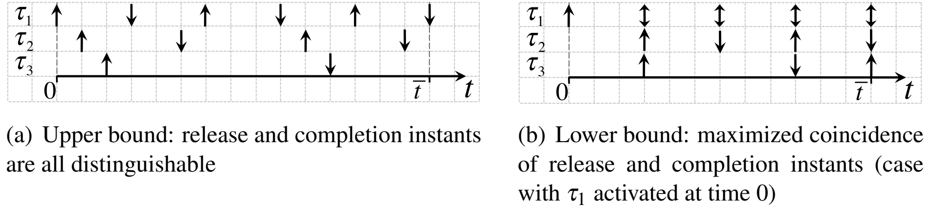
$$\mathsf{equals}(x,y) = \begin{cases} 1, & \text{if } x = y, \\ 0, & \text{otherwise} \end{cases}. \tag{2.48}$$

We recall that it is sufficient to test schedulability of $\mathscr{T} = \{\tau_1, \dots, \tau_n\}$ over time interval $[0, \bar{t}]$, with $\bar{t}$ defined by (2.34):

$$\bar{t} = \max_{\ell=1,\dots,m} C_\ell + \sum_{i=m+1}^{n} D_i. \tag{2.49}$$

Let p denote the number of scheduler invocations over time interval $[0, \bar{t}]$. The value of p can be bounded by

$$\underline{p} \leq p \leq \bar{p}$$



(a) Upper bound: release and completion instants are all distinguishable

(b) Lower bound: maximized coincidence of release and completion instants (case with τ_1 activated at time 0)

Figure 2.13: Bounds for the number of scheduler invocations p. Upward arrow denote job release, downward arrow denotes job completion

with

$$\underline{p} = \min_{i=1,\ldots,n} \left\lceil \frac{\bar{t}}{P_i} \right\rceil \qquad \text{and} \qquad \bar{p} = 2\sum_{i=1}^{n} \left\lceil \frac{\bar{t}}{P_i} \right\rceil \tag{2.50}$$

In the equation above, the upper bound $\bar{p}$ originates the execution scenario depicted in Fig. 2.13(a), when release and completion instants for jobs of $\tau_1,\ldots,\tau_n$ are all distinguishable (in other words, none of them coincides to any other), yielding the maximum number of scheduler invocations. Instead, the lower bound $\underline{p}$ originates the scenario depicted in Fig. 2.13(b), when the number of coinciding release and completion instants for $\tau_1,\ldots,\tau_n$ is maximized, minimizing the number of scheduler invocations.

It follows that a set of sporadic tasks $\mathscr{T} = \{\tau_1,\ldots,\tau_n\}$ is schedulable by GFP upon m processors, if and only if the following constraint programming problem is infeasible:

$$\begin{aligned} \exists \quad & p \in \{\underline{p},\ldots,\bar{p}\}, \quad (t_1,\ldots,t_p) \in \mathbb{R}^p, \quad R \in \mathbb{L}_R, \quad S \in \mathbb{L}_S, \quad F \in \mathbb{L}_F \\ & \exists \quad i = 1,\ldots,n, \quad j = 1,\ldots,p-1, \quad k = j+1,\ldots,p \\ & \longrightarrow \qquad \mathsf{h}_i(j,k)\,(t_k - t_j) \leq D_i, \end{aligned} \tag{2.51}$$

with its parameters defined as follows:

(a) $p \in \{\underline{p},\ldots,\bar{p}\}$ is the number of time instants, at which scheduling decisions happen, with $\underline{p}$, $\bar{p}$ defined by (2.50), and $t_1,\ldots,t_p$ represent time instants, when scheduling decisions are made, with $t_j < t_{j+1}$.

(b) $R = [r_{i,j}]$ is a binary matrix, representing a release sequence, with $i = 1,\ldots,n$ and $j = 1,\ldots,p$, such that element $r_{i,j} = 1$ if τ_i releases a job at time t_j, and $r_{i,j} = 0$ otherwise. $\mathbb{L}_R$ denotes all legal matrices R, defined by

$$R \in \mathbb{L}_R \quad \Longleftrightarrow \quad \left[\begin{array}{l} \begin{cases} \Delta t_{i,j} = 0 \\ r_{i,j} \in \{0,1\} \end{cases} \\ \begin{cases} \Delta t_{i,j} > 0 \\ r_{i,j} = 1 \ \Rightarrow \ \Delta t_{i,j} \geq P_i \end{cases} \end{array} \right. , \quad i = 1,\ldots,n, \quad j = 1,\ldots,p, \tag{2.52}$$

where $\Delta t_{i,j} = 0$ if τ_i has not released any job prior to time t_j, and $\Delta t_{i,j} > 0$ represents time

passed since the previous release of τ_i by time t_j, defined by

$$\Delta t_{i,j} = \sum_{k=1}^{j-1} r_{i,k} \left(1 - \sum_{\ell=1}^{k-1} r_{i,\ell}\right)_0 \left(t_j - \left[t_k + \left(\sum_{\ell=1}^{j-1} r_{i,\ell} - 1\right) P_i\right]\right).$$

According to definition (2.52), R is legal if and only if time separation between any consecutive releases of each τ_i is P_i.

(c) $S = [s_{i,j}]$ is a binary matrix, representing a resource schedule, such that $s_{i,j} = 1$ if a processor is allocated to τ_i over time interval $[t_j, t_{j+1})$, otherwise $s_{i,j} = 0$. $\mathbb{L}_E$ denotes all legal GFP schedules for $\mathscr{T}$, defined by

$$E \in \mathbb{L}_E \quad \Longleftrightarrow \quad \begin{cases} s_{i,j} = 1 \;\Leftrightarrow\; \begin{cases} \sum_{\ell=1}^{j} r_{i,\ell} > \sum_{\ell=1}^{j} f_{i,\ell} & (E1) \\ \sum_{\ell=1}^{i-1} s_{\ell,j} \leq m-1 & (E2) \end{cases} \\ s_{i,j} = 0, \quad \text{otherwise} \\ \qquad\qquad i = 1,\ldots,n, \quad j = 1,\ldots,p \end{cases}, \tag{2.53}$$

meaning that a processor is allocated to τ_i at time t_j if i) τ_i has a pending job at time t_j (corresponds to constraint (E1)), and ii) the number of processors allocated to higher priority tasks does not exceed $m-1$ (corresponds to constraint (E2)).

(d) $F = [f_{i,j}]$ denotes a binary matrix, representing a finishing sequence, such that element $f_{i,j} = 1$ if some job of τ_i is completed at time t_j, and $f_{i,j} = 0$ otherwise. $\mathbb{L}_C$ denotes all legal matrices F for given R, defined by

$$F \in \mathbb{L}_F \quad \Longleftrightarrow \quad \begin{cases} f_{i,j} = 1 \quad \Leftrightarrow \quad \mathsf{supply}_i(j) = C_i \\ f_{i,j} = 0 \quad \Leftrightarrow \quad \mathsf{supply}_i(j) < C_i \\ \qquad i = 1,\ldots,n, \quad j = 1,\ldots,p \end{cases}, \tag{2.54}$$

where $\mathsf{supply}_i(j)$ represents the amount of resource allocated to the pending job of τ_i, which has the earliest release time compared to other pending jobs of τ_i at time t_{j-1} (if any), from its release time to time t_j, computed by

$$\mathsf{supply}_i(j) = \sum_{k}^{j-1} \mathsf{f}_i(k,j)\, \mathsf{s}_i(k,j),$$

where

(a) $\mathsf{f}_i(k,j) = 1$ if τ_i releases a job at time t_k, which is incomplete at time t_{j-1}, and has the earliest deadline compared to other τ_i pending jobs at time t_{j-1}, otherwise $\mathsf{f}_i(k,j) = 0$,

defined by

$$\mathsf{f}_i(k,j) = \mathsf{r}_{i,k} \text{ equals}\left(\sum_{\ell=1}^{k-1} r_{i,\ell},\ \sum_{\ell=1}^{j-1} f_{i,\ell}\right).$$

(b) function $\mathsf{s}_i(k,j)$ computes the amount of resource allocated to a job of τ_i, released at time t_k, over time interval $[t_k, t_j)$, defined by

$$\mathsf{s}_i(k,j) = \sum_{\ell=k}^{j-1} \mathsf{e}_{i,\ell}\,(t_{\ell+1} - t_\ell) \times \text{equals}\left(\sum_{p=1}^{j-1} \mathsf{r}_{i,p},\ \sum_{p=1}^{\ell} \mathsf{c}_{i,p}\right).$$

(e) $\mathsf{h}_i(j,k)$ is a boolean function, such that $\mathsf{h}_i(j,k) = 1$ if and only if τ_i releases some job at time t_j, which is respectively completed at time t_k, defined by

$$\mathsf{h}_i(j,k) = r_{i,j}\, f_{i,k} \text{ equals}\left(\sum_{\ell=1}^{j} r_{i,\ell},\ \sum_{\ell=1}^{k} f_{i,\ell}\right). \tag{2.55}$$

Feasibility of the CP problem (2.51) can be checked by a CP solver, such as GECODE or IBM CP Optimizer. In case the problem (2.51) has no feasible solution, then task set $\mathscr{T}$ is schedulable, otherwise $\mathscr{T}$ is unschedulable.

2.8 Summary

We evaluated the pessimism of the state-of-the-art sufficient schedulability tests for GFP, and confirmed its relevance. Then, to reduce the limitations of the existing exact tests - these are high computation time and memory consumption -, we derived a set of improved exact tests, exploring the idea of pruning of the analyzed state space.

The first test extended the work of Bonifaci and Marchetti-Spaccamela (2012), by using a set of pruning methods for shrinking the state space. The evaluation confirmed our test to be the fastest and the least memory consuming compared to all other existing exact tests, although the exponential time and polynomial space complexity remains, as for Bonifaci's test.

Another test is designed for continuous-time scheduling, e.g. an event-driven scheduler, which is opposed to a tick-driven scheduler. Although has not been yet implemented, such a test is expected to have a significantly lower time and space complexity, due to usage of polynomial-time linear programming solvers, rather than using enumeration-like methods only.

The third test we proposed is formulated by means of a constraint programming (CP) problem, with the use of logical constraints. The resulted CP problem can be solved by using a CP solver, such as GECODE or IBM CP Optmizer.

The derived models are easily adjustable for more general task models, such as DAG tasks, or hierarchical scheduling.

Chapter 3

Schedulability Analysis of Compositional Real-time Systems

Another problem we consider is the schedulability analysis of compositional multiprocessor real-time systems. Despite of hardness of such a problem, we succeed to provide a solving algorithm of a linear runtime complexity, in an average case, thanks to derived tight pruning methods for the solution search space.

3.1 Motivation

Reusing application code is driven by the need to shorten the overall design time, and typically software components are developed in isolation, possibly by different developers. During the integration phase, all components are bound to the same hardware platform. Clearly, the integration must be performed in such a way that the properties of components are preserved even after the composition is made.

In real-time systems, the key property that has to be preserved during the integration phase is time predictability: a real-time application (or component) that meets all its deadlines when designed in isolation should also meet all deadlines when it is integrated with other applications on the same hardware platform. This property is often guaranteed by introducing an *interface* between the application and the hardware platform. Then the application is guaranteed over the interface, and the hardware platform must provide a *virtual platform* that conforms with the interface - a compliant virtual platform. The scheduling problem over a virtual platform is often called a *hierarchical scheduling* problem. In fact, each application task itself may contain another entire application in a hierarchical fashion.

The benefit of using an interface-based approach is significant. During the design phase the interface of an application is computed such that all timing requirements of the application are met. Then, during the integration phase the interfaces of all applications are bound to the same hardware platform. As a result, the interface allows to hide an internal complexity of an individual application, and this property is essential in the development of large-scale real-time systems.

Typically, interfaces, allowing the composition of real-time applications, specify details about the amount of resource that has to be provided by a compliant virtual platform. This information can be described with a varying degree of detail. For example, a very simple interface for a virtual processor can be just a fraction of the allocated time.

With the broad diffusion of multiprocessors, hierarchical scheduling problems have recently started to be considered over hardware platforms that provide a concurrent resource supply. The formulation of interface models for multiprocessors, however, requires the introduction of a new dimension: the *degree of concurrency*. This additional characteristic of the interface makes the problem to be addressed more challenging.

The problem in selecting the appropriate interface model is to find the best trade-off between accuracy and simplicity of the interface. A simple interface is intuitive and easy to use, but it tends to cause a significant pessimism in the resource abstraction. On the other hand, an accurate interface minimizes the pessimism, but is more complex in use, and it can be very difficult to compute. In this paper we propose a simple interface that is a generalization of the one previously proposed by Shin et al. (2008). Our novel approach keeps the simplicity of that interface while reducing significantly the pessimism in terms of the needed resource.

3.2 Contributions

To analyze schedulability of compositional multiprocessor systems, we first introduce the generalized multiprocessor periodic resource (GMPR) interface. GMPR generalizes the MPR model of Shin et al. (2008), as well as other periodic interface models, reducing their pessimism while remaining to be simple.

To analyze schedulability over GMPR, we determine the worst-case resource allocation patterns over it, allowing both, identical or different periods for each virtual processor. As a schedulability test, we reuse the sufficient test of Bini et al. (2009). We first improve this test by minimizing its runtime, and then, based on it, we compute GMPR by solving a set of non-convex mixed-integer optimization problems.

To make solving of such complex optimization problems possible, we derive a set of tight pruning constraints for the solution search space, so that an adequate optimization solver could quickly find an optimal solution. In the end, we succeed to reduce the exponential runtime complexity of GMPR computation to just linear.

To confirm a reduced pessimism of GMPR, we have implemented the solution in the Matlab environment.

3.3 Related works

The problem of composing real-time applications is certainly not new. There actually have been numerous contributions in this area. Being fully aware of the impossibility to provide a full coverage of the topic, we describe in this section the works that, to our best knowledge, are more related

to ours.

One of the first contributions to address the isolation of applications using *resource reservations* was published in (Parekh and Gallager, 1993). In that paper the authors introduced the Generalized Processor Sharing (GPS) algorithm to share a fluid resource according to a set of weights. Mercer et al. (1994) proposed a more realistic approach where a resource can be allocated based on a required budget and period. Later on, Stoica et al. (1996) introduced the Earliest Eligible Virtual Deadline First (EEVDF) for sharing the computing resource, and Deng and Liu (1997) achieved the same goal by introducing a two-level scheduler (using EDF as a global scheduler) in the context of multi-application systems. Kuo and Li (1999) extended the approach to a Fixed Priority global scheduler. Kuo et al. (2000) extended their own work (Kuo and Li, 1999) to multiprocessors. However, in those approaches the authors made very stringent assumptions such as not considering task migration and restricting to period harmonicity. Those assumptions restrict the applicability of the proposed solution.

Moir and Ramamurthy (1999) proposed a hierarchical approach, where a set of P-fair tasks can be scheduled within a time partition provided by another P-fair task (called "supertask") acting as a server. However, the solution often requires the weight of the supertask to be higher than the sum of the weights of the served tasks (Holman and Anderson, 2006).

Many independent works proposed to model the service provided by a uni-processor through a supply function. Feng and Mok (2002) introduced the bounded-delay resource partition model. Almeida et al. (2002) provided timing guarantees for both synchronous and asynchronous traffic over the FTT-CAN protocol by using hierarchical scheduling. Lipari and Bini (2003) derived the set of virtual processors that can feasibly schedule a given application. Shin and Lee (2003) introduced the periodic resource model also deriving a utilization bound. Easwaran et al. (2007) extended this model allowing the server deadline to be different from its period. Fisher and Dewan (2009) proposed an approximation algorithm to test the schedulability of a task set over a periodic resource.

More recently, some authors have addressed the problem of specifying an interface for applications executed upon multiprocessor systems, providing appropriate tests to verify schedulability of applications over that interface.

One of such works is described in (Leontyev and Anderson, 2008), where the authors proposed to use only the overall bandwidth requirement w as interface for soft real-time applications. The authors propose to allocate a bandwidth requirement of w onto $\lfloor w \rfloor$ dedicated processors, plus an amount of $w - \lfloor w \rfloor$ provided by a periodic server globally scheduled onto the remaining processors. An upper bound of the tardiness of tasks scheduled on such an interface was provided.

Shin et al. (2008) proposed the multiprocessor periodic resource model (MPR) that specifies a period, a budget and maximum level of parallelism of the resource provisioning. Khalilzad et al. (2012) later extended the MPR model, relaxing the assumption of fully synchronized virtual processors. Since our work is a generalization of the MPR, in Section 3.4.2 we describe the MPR in greater details.

Chang et al. (2008) proposed to partition the resource available from a multiprocessor by a static periodic scheme. The amount of resource is then provided to the application through a contract specification.

Bini et al. (2009) proposed the Parallel Supply Function (PSF) interface of a virtual multiprocessor. This interface is designed to tightly capture the amount of resource provided by a virtual platform for very general supply mechanisms, which are not necessarily periodic. In their approach the authors do not reason on how to compute the interface parameters that guarantee the schedulability of a real-time application.

Lipari and Bini (2010) described an entire framework for composing real-time applications running over a multiprocessor. However, their proposed interface was extremely trivial.

Burmyakov et al. (2012) extended the multiprocessor periodic resource model (MPR) by specifying the minimal budgets for each level of parallelism. However, the assumption of integer budget values made the problem to compute an interface hardly tractable, even for a task set with a low utilization.

3.4 Background on compositional scheduling

In the past, there have been some proposals for multiprocessor interfaces. This section illustrates three of them (Leontyev and Anderson, 2008; Shin et al., 2008; Bini et al., 2009). The interfaces are ordered by their increasing complexity and, consequently, by increasing accuracy of the guarantee test for applications running over the interface.

3.4.1 The multiprocessor bandwidth interface (MBI)

Leontyev and Anderson (2008) proposed to use only the overall bandwidth requirement w (using their original notation) as an interface for soft real-time tasks. Being a multiprocessor interface, it is well acceptable to have $w > 1$. To schedule a task set, the authors proposed to allocate a bandwidth requirement of w onto $\lfloor w \rfloor$ fully dedicated processors, plus the bandwidth of $w - \lfloor w \rfloor$ provided by a periodic server globally scheduled onto the remaining processors (see Fig. 3.1).

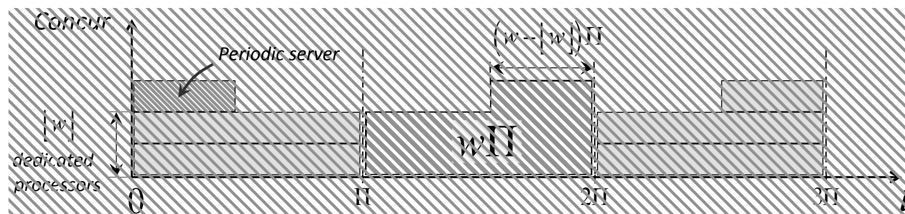


Figure 3.1: MBI: the worst-case resource pattern

We refer the interface model of Leontyev and Anderson (2008) as the multiprocessor bandwidth interface (MBI) and denote it as

$$\langle w, \Pi \rangle,$$

where w is the interface bandwidth and Π is the server period. Initially designed for soft real-time tasks, the MBI model can easily be extended for hard real-time systems. The advantage of the MBI is its simplicity and the reduced pessimism in the resource abstraction compared to many other existing models.

At the same time, there is a strong limitation of the MBI model as it requires $\lfloor w \rfloor$ fully dedicated processors. In a general case of the compositional scheduling, such a requirement cannot be always guaranteed by a virtual execution platform, for extended periods of time. To overcome this limitation, other different interface models have been introduced, as described in the next sections.

3.4.2 The multiprocessor periodic resource model (MPR)

The multiprocessor periodic resource model (MPR) (Shin et al., 2008) is another simple resource abstraction. Its definition is given below.

Definition 3. *A Multiprocessor Periodic Resource model (MPR) is modeled by a triplet*

$$\langle \Pi, \Theta, m \rangle,$$

where Π is the time period and Θ is the minimal resource supply provided within each time interval $[k\Pi, (k+1)\Pi)$, with $k \in \mathbb{N}_0$, by at most m processors at a time. Often we also say that m is the concurrency *(or the* degree of parallelism*) of the interface. The* utilization *of a MPR interface is the ratio $\frac{\Theta}{\Pi}$.*

Since a MPR interface fixes only the aggregated parameters Π, Θ and m of the supply pattern, any feasible allocation of Θ resource units per time period Π with a parallelism m should preserve the schedulability of the underlying task set. It is then necessary to find the worst-case resource allocation for the MPR. Generalizing the result of Shin et al. (2008), derived for a case of integer Θ, the worst-case scenario for an arbitrary Θ is the one depicted in Fig. 3.2, where time instant 0 denotes the beginning of the worst-case interval. Note that in the MPR case the contribution of each processor to the interface is Θ/m every period Π.

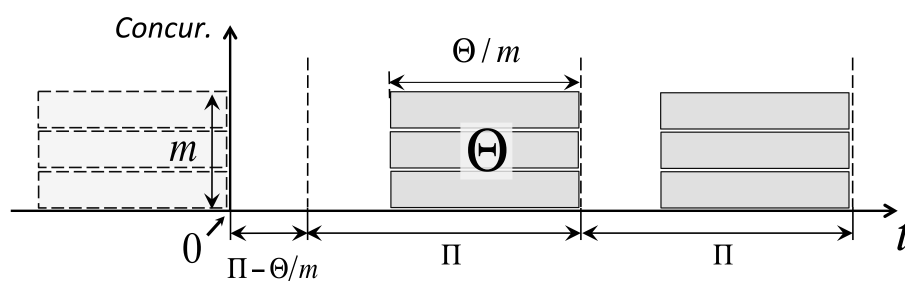


Figure 3.2: MPR: the worst-case resource pattern. Instant 0 denotes the beginning of the worst-case interval.

Table 3.1: An example of a task set.

i	C_i	T_i	D_i
1	1	30	30
2	4	40	40
3	11	50	50
4	15	60	60

3.4.3 Comparison of MBI and MPR resource interfaces

The MBI model dominates MPR in terms of overall resource required to schedule an application: over the same time interval, MBI requires at most as much resource as MPR. However, unlike MBI, an MPR interface can be also provided over a platform in which the processors are not fully available (possibly due to the coexistence with other applications already consuming resource). In fact, by increasing the interface parallelism m, the requirement Θ/m on each processor decreases, making it possible to fit an interface on partially available platforms.

We illustrate this by an example. Consider a task set with the parameters reported in Table 3.1, to be scheduled by global EDF (GEDF) over a virtual platform. To compute interfaces, we apply the schedulability test of Lipari and Bini (2010), which is described in details later in Section 3.6. By setting the server period to $\Pi = 20$, we determine that the minimal MBI interface, guaranteeing the schedulability of the task set, requires 26 resource units every Π, while the MPR of the same concurrency $m = 2$ requires at least 30.8 units (see Fig. 3.3).

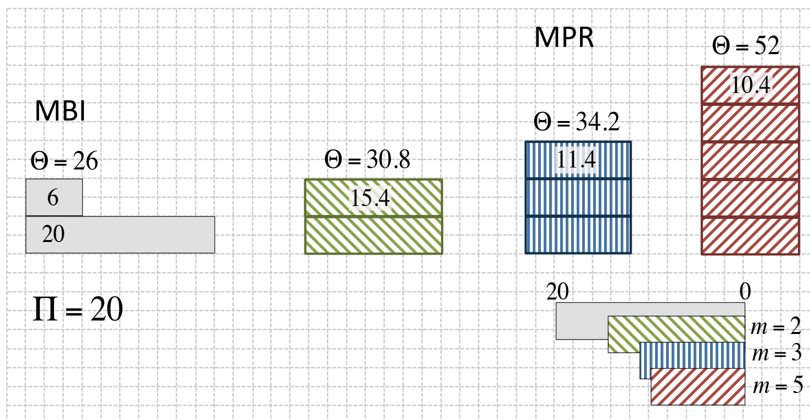


Figure 3.3: Comparison of MBI and MPR resource interfaces

Let us now increase the MPR concurrency to $m = 3$. We immediately obseve a reduction of resource to be provided by each virtual processor, from 15.4 to 11.4 units. For $m = 5$, the resource fraction decreases further to 10.4. Notice, however, that the overall resource Θ increases with m.

3.4.4 The parallel supply function (PSF)

The parallel supply function (PSF) was proposed by Bini et al. (2009) to characterize the resource allocation in hierarchical systems executed upon a multiprocessor platform. This interface is designed to tightly capture the amount of resource provided by a virtual platform for very general supply mechanisms, which are not necessarily periodic. As a drawback it is certainly quite complicated to be handled. Without entering into all the details of the definition (that can indeed be found in (Bini et al., 2009)), we recall here the basic concepts.

Definition 4. *The Parallel Supply Function (PSF) interface of a multiprocessor resource is composed by the set of functions $\{Y_k\}_{k=1}^m$, where m is the number of virtual processors and $Y_k(t)$ is the* minimum *amount of resource provided in* any *interval of length t with a parallelism of* at most *k. The function $Y_k(t)$ is called the* level-*k* parallel supply function.

To clarify this definition we propose an example. Consider that in the interval $[0,11]$ the resource is provided by three processors according to the schedule drawn in gray in Fig. 3.4.

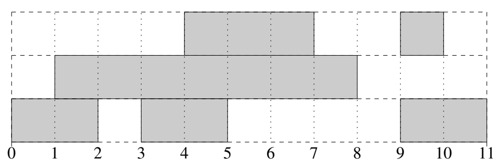


Figure 3.4: An example of a resource allocation scenario for PSF computation

In this case $Y_1(11) = 10$ because there is always at least one processor available in $[0,11]$ except in $[8,9]$. Then $Y_2(11) = 16$; that is found by summing up all the resources except one with parallelism 3 (provided only in $[4,5]$). Finally, $Y_3(11) = 17$; that is achieved by summing all the resources provided in $[0,11]$. In general, the parallel supply functions are also computed by sliding the time window of length t and by searching for the most pessimistic scenario of resource allocation. This minimization is somehow equivalent to the one performed on uni-processor hierarchical scheduling (Feng and Mok, 2002; Lipari and Bini, 2003; Shin and Lee, 2003) for computing the supply function of a virtual resource.

Since the PSF can be computed for any possible resource allocation scheme, it is possible to compute it also for the MPR interface. The computation of the PSF interface $\{Y_k\}_{k=1}^m$ of a MPR enables the adaptation of schedulability tests developed over a PSF interface to a MPR interface. More details about the schedulability test will be provided in Section 3.6.

3.5 The generalized multiprocessor periodic resource (GMPR) model

The main drawback of the MPR interface is that it may require more computational capacity than needed, and therefore it has an undesirable level of pessimism in terms of resource allocation. Consider the task set with the parameters as depicted in Table 3.2, to be scheduled by global EDF (GEDF) over the MPR interface. In that table, for each task we provide its execution time, C_i, its period, T_i, and its deadline, D_i.

Table 3.2: An example of a task set.

i	C_i	T_i	D_i
1	6	40	40
2	13	50	50
3	29	60	60
4	27	70	70

After setting the period of the interface $\Pi = 15$, we compute a MPR interface $\langle \Pi, \Theta, m \rangle$ that can guarantee the task set. To check the schedulability, we reuse the PSF-based test proposed by Bini et al. (2009) (see Section 3.6 for details). Based on this test, we determine that the minimum feasible value of resource units to guarantee the schedulability is $\Theta = 39$. Notice that there is quite a significant gap between the utilization of the interface $\frac{\Theta}{\Pi} = 2.6$ and the utilization of the task set $\sum_i \frac{C_i}{T_i} = 1.28$.

As we will show in greater detail in the next sections, our proposed interface requires only 34 resource units per period, meaning that it has a utilization of $\frac{34}{15} = 2.267$ for the given example.

3.5.1 Definition

The main reason for the pessimism of the MPR is that the worst-case of the supply (Fig. 3.2) must be very conservative, if the only information in the interface is that an overall budget Θ is provided every Π. We propose to rectify this problem, as described below.

Definition 5. *We define the Generalized Multiprocessor Periodic Resource interface model (GMPR) as*

$$\langle \Pi, \{\Theta_1, \ldots, \Theta_m\} \rangle,$$

where Π is the time period and Θ_k is the minimal resource supply provided within each time interval $[\ell\Pi, (\ell+1)\Pi)$, $\ell \in \mathbb{N}_0$, with a degree of parallelism of at most k. The values of Θ_k must satisfy the following constraints for any $k = 1, \ldots, m$ (for convenience we denote $\Theta_k = 0,\ k \leq 0$):

$$\begin{aligned} &0 \leq \Theta_k - \Theta_{k-1} \leq \Pi \\ &\Theta_{k+1} - \Theta_k \leq \Theta_k - \Theta_{k-1}. \end{aligned} \tag{3.1}$$

We assume that the interface parameters Π and $\Theta_1, \ldots, \Theta_m$ belongs to $\mathbb{R}$.

The "degree of parallelism" of a resource supply at a time instant t, is the number of processors providing the resource at that instant. For example, an application which may have at most ℓ threads in parallel will not ever benefit from having a resource provided by $\ell + 1$ processors simultaneously. Hence, for such an application, it does not make sense to have $\Theta_{\ell+1}$ strictly larger than Θ_ℓ, since the extra amount of resource $\Theta_{\ell+1} - \Theta_\ell$ is provided at a too high parallelism that the application never exhibits.

The motivation for the constraints in Definition 5 is the following:

- $\Theta_k \geq \Theta_{k-1}$, because the overall supply at higher parallelism cannot decrease;
- $\Theta_k - \Theta_{k-1} \leq \Pi$, because the increment of supply at parallelism k (that is $\Theta_k - \Theta_{k-1}$) cannot exceed the length of the period;
- $\Theta_{k+1} - \Theta_k \leq \Theta_k - \Theta_{k-1}$, because the increment of supply at parallelism $k+1$ (that is $\Theta_{k+1} - \Theta_k$) should not exceed the increment of supply at parallelism k (that is $\Theta_k - \Theta_{k-1}$). Otherwise some of the supply provided at parallelism $k+1$ must instead be available at parallelism k.

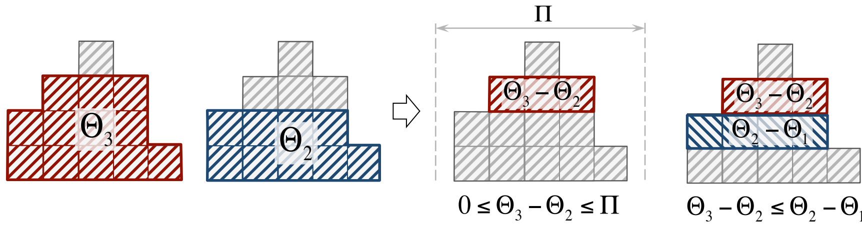


Figure 3.5: GMPR definition: the graphical interpretation

Fig. 3.5 illustrates an example of a resource supply over a GMPR interface with $\Pi = 6$, $\Theta_1 = 5$, $\Theta_2 = 9$, and $\Theta_3 = 12$.

A valid GMPR interface should guarantee the schedulability of a task set: any resource allocation compliant with the GMPR specification has to guarantee that all task deadlines are met.

The proposed GMPR interface model generalizes both MPR and MBI. In fact, a MPR interface $\langle \Pi, \Theta, m \rangle$ is equivalent to a GMPR $\langle \Pi, \{\Theta_1, \ldots, \Theta_m\} \rangle$ with

$$\Theta_k = \frac{k}{m}\Theta, \qquad k = 1, \ldots, m,$$

and a MBI interface $\langle w, \Pi \rangle$ is equivalent to a GMPR with

$$\Theta_k = k\Pi, \qquad k = 1, \ldots, \lfloor w \rfloor$$
$$\Theta_{\lceil w \rceil} = w\Pi.$$

3.5.2 Parallel supply functions of GMPR

To borrow the schedulability tests developed over the PSF interface (Bini et al., 2009), we compute the parallel supply functions $\{Y_k(t)\}_{k=1,\ldots,m}$ for the GMPR specification.

Burmyakov et al. (2012) proposed to compute the PSF using a classical approach in hierarchical scheduling. In that work the authors considered the worst-case scenario of the resource supply (depicted in Fig. 3.6) and defined $\mathsf{supply}_k(t)$ as the amount of resource available in $[0,t]$ by at most k concurrent processors (see Fig. 3.6). Then, the PSF $Y_k(t)$ was computed as

$$Y_k(t) = \min_{t_0 \in T} \left(\mathsf{supply}_k(t + t_0) - \mathsf{supply}_k(t_0) \right),$$

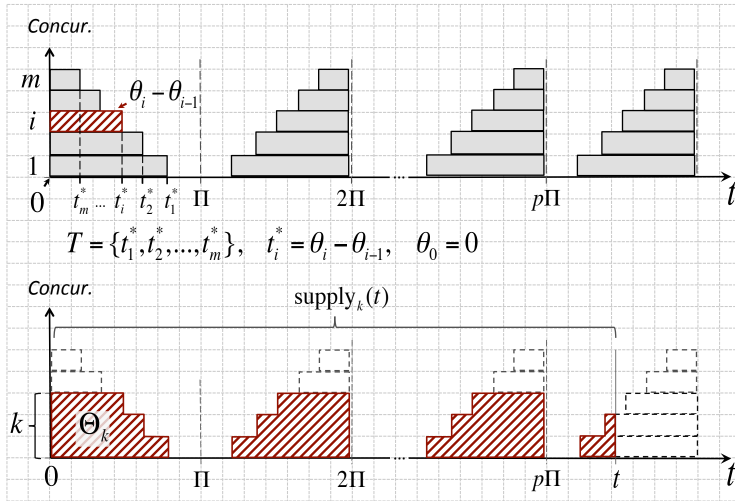


Figure 3.6: GMPR: the worst-case resource pattern (top) and the definition of the supply$_k(t)$ function (bottom) proposed by Burmyakov et al. (2012).

with $T = \{\Theta_i - \Theta_{i-1} | i = 1, \ldots, k\}$ being the set of time instants at which the supply by some processor ends.

Instead of the above mentioned approach, we now propose a significantly more efficient method to compute the Parallel Supply Functions $Y_k(t)$. We stress that this method is also applicable to the classical problems of hierarchical scheduling over a single processor (Lipari and Bini, 2003; Shin and Lee, 2003), as PSF is a generalization of the uni-processor supply function.

To compute the PSF $Y_k(t)$, let us first introduce an auxiliary function $s_k(t)$ over $t \in [0, \Pi]$. We define $s_k(t)$ as the overall amount of resource provided over the pattern of Fig. 3.7, in a time interval $[0, t]$. The function $s_k(t)$ has the property, formulated in the next lemma.

Lemma 1. *Let $s_k : [0, \Pi] \to \mathbb{R}$ be defined as*

$$s_k(t) = \sum_{i=1}^{k} (t - (\Pi - (\Theta_i - \Theta_{i-1})))_0 . \tag{3.2}$$

Then, for any values $t_1, t_2 \in [0, \Pi]$, we have

$$s_k(t_1) + s_k(t_2) \geq 2\, s_k\left(\frac{t_1 + t_2}{2}\right). \tag{3.3}$$

Proof. Consider the resource allocation over the time interval $[t_1, t_2]$ of Fig. 3.7. Time instant $t = \frac{t_1+t_2}{2}$ is the middle of this interval. Due to the alignment of the resource blocks to the right side, the resource in $[t_1, \frac{t_1+t_2}{2}]$ does not exceed the resource in $[\frac{t_1+t_2}{2}; t_2]$. It follows that

$$s_k\left(\frac{t_1 + t_2}{2}\right) - s_k(t_1) \leq s_k(t_2) - s_k\left(\frac{t_1 + t_2}{2}\right),$$

what leads us to (3.3). □ □

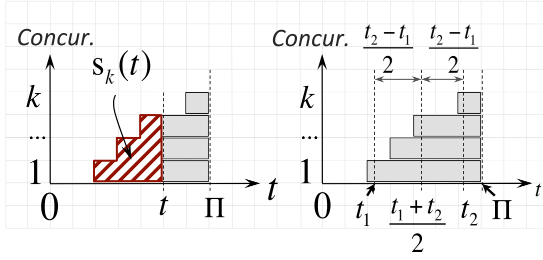


Figure 3.7: Properties of GMPR supply function $s_k(t)$

The next theorem determines the worst-case scenarios of the resource supply which are then used to compute $Y_k(t)$.

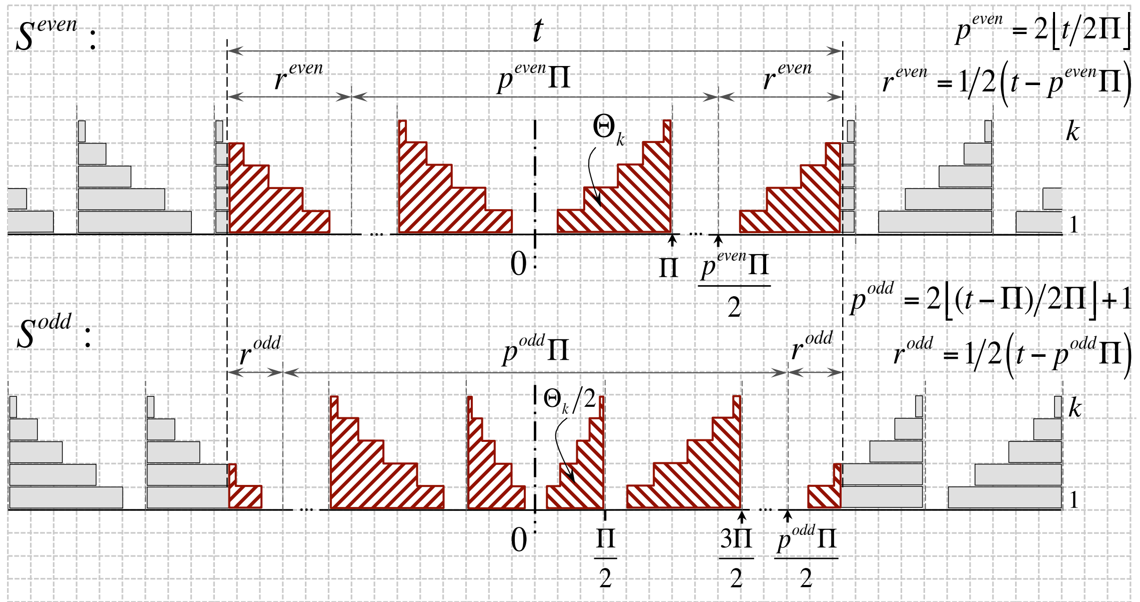


Figure 3.8: GMPR: the worst-case resource patterns S^{even} and S^{odd}

Theorem 2. *The worst-case amount of resource provided over a GMPR interface $\langle \Pi, \{\Theta_1, \dots, \Theta_m\} \rangle$ in an arbitrary time interval of length t is the minimum among the resources provided in $[-\frac{t}{2}, \frac{t}{2}]$ by any of the two patterns S^{even} and S^{odd} depicted in Fig. A.4.*

The proof of Theorem 2 is provided in Appendix A.3.

Theorem 2 determines that the worst-case pattern for the resource supply of a GMPR interface is either S^{odd} or S^{even}. The next corollary uses such a result to compute the PSF of a GMPR interface.

Corollary 3. *The PSF function $Y_k(t)$ for GMPR is computed as*

$$Y_k(t) = \min\left(Y_k^{\text{even}}(t), Y_k^{\text{odd}}(t)\right), \tag{3.4}$$

where Y_k^{even} and Y_k^{odd} denote the resource provided by the patterns S^{even} and S^{odd} depicted in Fig. A.4, computed as

$$Y_k^{\mathsf{even}}(t) = p^{\mathsf{even}}\,\Theta_k + 2\sum_{i=1}^{k}(r^{\mathsf{even}} - \Pi + \Theta_i - \Theta_{i-1})_0 \tag{3.5}$$

$$p^{\mathsf{even}} = 2\left\lfloor \frac{t}{2\Pi} \right\rfloor \tag{3.6}$$

$$r^{\mathsf{even}} = \frac{1}{2}(t - p^{\mathsf{even}}\,\Pi) \tag{3.7}$$

and

$$Y_k^{\mathsf{odd}}(t) = p^{\mathsf{odd}}\,\Theta_k + 2\sum_{i=1}^{k}\left(r^{\mathsf{odd}} - \Pi + \Theta_i - \Theta_{i-1}\right)_0 \tag{3.8}$$

$$p^{\mathsf{odd}} = 2\left\lfloor \frac{t-\Pi}{2\Pi} \right\rfloor + 1 \tag{3.9}$$

$$r^{\mathsf{odd}} = \frac{1}{2}\left(t - p^{\mathsf{odd}}\,\Pi\right). \tag{3.10}$$

As an example, in Fig. 3.9 we depict supply functions $\{Y_1(t), \ldots, Y_4(t)\}$ for GMPR $\langle 7, \{6, 11, 15, 17\}\rangle$. At the bottom of the figure we also depict the worst-case resource patterns orginating these functions.

3.5.3 GMPR supply bounds

We now propose the lower and the upper bounds for GMPR supply function $Y_k(t)$. These bounds will be exploited later in Section 3.7.3, in order to reduce computation time for GMPR interface.

The supply functions $Y_k^{\mathsf{even}}(t)$, $Y_k^{\mathsf{odd}}(t)$ defined by equations (3.5), (3.8) can be equally expressed as

$$\begin{aligned} Y_k^{\mathsf{even}}(t) &= p^{\mathsf{even}}\,\Theta_k + 2s_k(r^{\mathsf{even}}) \\ Y_k^{\mathsf{odd}}(t) &= p^{\mathsf{odd}}\,\Theta_k + 2s_k(r^{\mathsf{odd}}), \end{aligned} \tag{3.11}$$

with $s_k(t)$ defined by (3.2), and p^{even}, r^{even}, p^{odd}, r^{odd} defined by (3.6), (3.7), (3.9), and (3.10), respectively.

We now observe that the function $s_k(t)$ can be lower bounded by the function $\underline{s}_k(t)$ defined as (see also Fig. 3.10)

$$s_k(t) \;\geq\; \underline{s}_k(t) = (\Theta_k - k(\Pi - t))_0\,. \tag{3.12}$$

Substituting (3.12) into (3.11), we derive the following lower bounds for $Y_k^{\mathsf{even}}(t)$, $Y_k^{\mathsf{odd}}(t)$ denoted as $\underline{Y}_k^{\mathsf{even}}(t)$, $\underline{Y}_k^{\mathsf{odd}}(t)$:

$$\underline{Y}_k^{\mathsf{even}}(t) = p^{\mathsf{even}}\,\Theta_k + 2\,(\Theta_k - k\,(\Pi - r^{\mathsf{even}}))_0 \tag{3.13}$$

$$\underline{Y}_k^{\mathsf{odd}}(t) = p^{\mathsf{odd}}\,\Theta_k + 2\left(\Theta_k - k\left(\Pi - r^{\mathsf{odd}}\right)\right)_0. \tag{3.14}$$

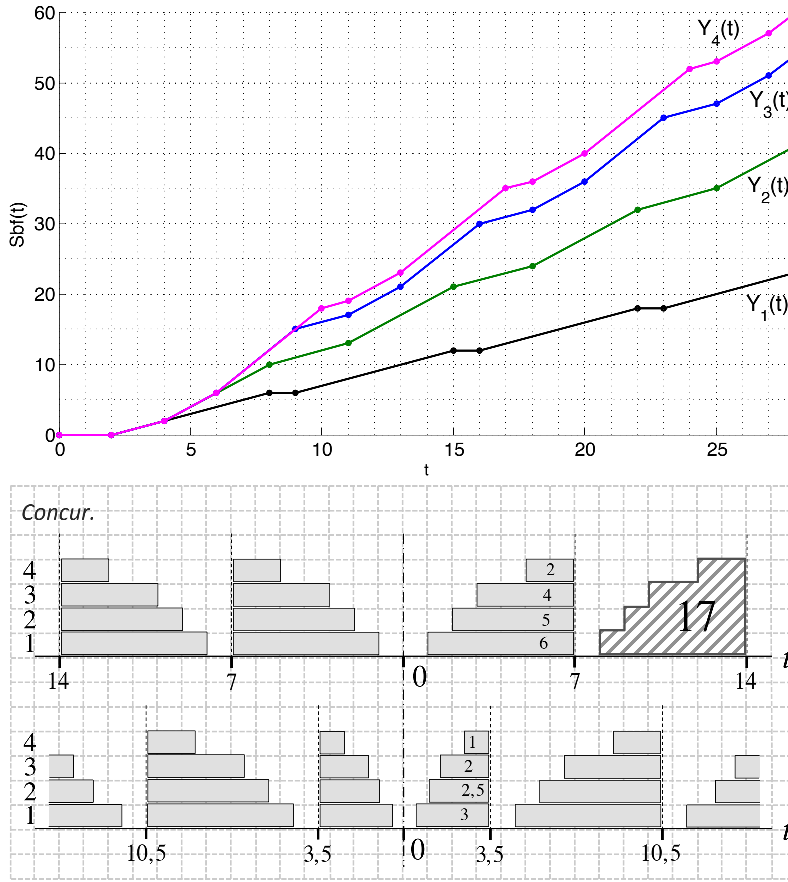


Figure 3.9: The PSF (top) and the worst-case supply patterns (bottom) of the GMPR interface $\langle 7, \{6, 11, 15, 17\}\rangle$. The bold points indicate the slope change of the PSF functions.

The bounds $\underline{Y}_k^{\text{odd}}(t)$, $\underline{Y}_k^{\text{even}}(t)$ are plotted in Fig. 3.11. Considering equation (3.4) and Fig. 3.11, we conclude that a valid lower bound for $Y_k(t)$ is $\underline{Y}_k(t)$ defined as

$$\underline{Y}_k(t) = \frac{\Theta_k}{\Pi} t - 2\frac{\Theta_k}{\Pi}\left(\Pi - \frac{\Theta_k}{k}\right). \tag{3.15}$$

The upper bound $\overline{Y}_k(t)$ for $Y_k(t)$ is derived in a similar way. First, we observe that the function $s_k(t)$ is upper bounded by the function $\bar{s}_k(t)$ depicted in Fig. 3.10. Then, substituting the expression for $\bar{s}_k(t)$ into (3.11), we derive the upper bounds $\overline{Y}_k^{\text{even}}(t)$, $\overline{Y}_k^{\text{odd}}(t)$ for $Y_k^{\text{even}}(t)$, $Y_k^{\text{odd}}(t)$, and in the

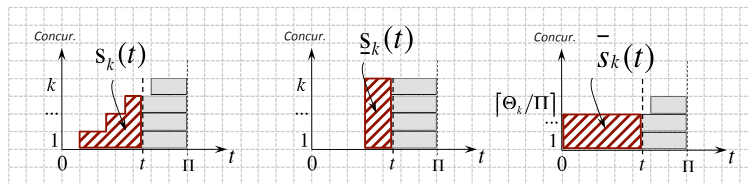


Figure 3.10: The lower and the upper bounds $s_k(t)$ (put underline), $\bar{s}_k(t)$ for GMPR supply function $s_k(t)$. The overall supply allocated over $[0;\Pi]$ is Θ_k.

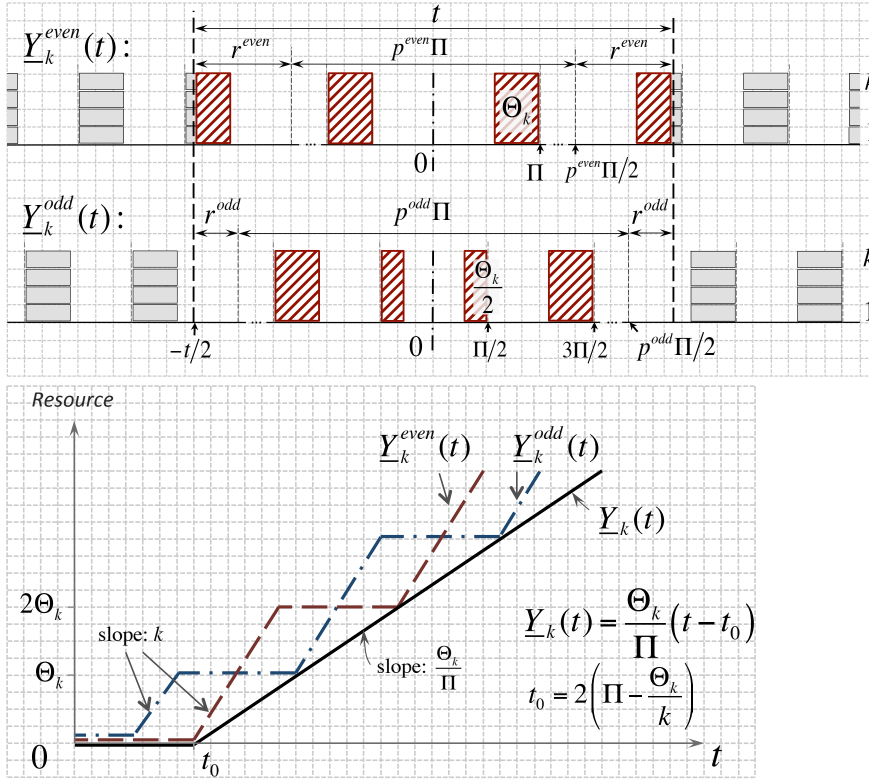


Figure 3.11: The lower bound for GMPR supply $Y_k(t)$.

end we determine that

$$\overline{Y}_k(t) = \frac{\Theta_k}{\Pi} t, \tag{3.16}$$

as

$$\overline{Y}_k(t) \geq \max\left(\overline{Y}_k^{\mathsf{even}}(t), \overline{Y}_k^{\mathsf{odd}}(t)\right).$$

3.5.4 Supply functions for GMPR with different processor periods

Until now we assumed that all virtual processors for GMPR have an identical replenishment period Π. Below we briefly show that Corollary 3 for the worst-case supply is easily extendable to GMPR with different replenishment periods for virtual processors.

Let us extend definition (5) of GMPR by allowing a different replenishment period for each virtual processor, that is

$$\mu_{\mathsf{ext}} = \langle \{\Pi_k, \Theta_k\}_{k=1}^{m} \rangle, \tag{3.17}$$

where Π_k denotes the period of the k-th virtual processor. Corollary 3 for such μ_{ext} is generalized as follows.

Corollary 4. *The PSF function $Y_k(t)$ for extended GMPR μ_{ext} is computed by*

$$Y_k(t) = \sum_{\ell=1}^{k} \min\left(y_\ell^{\mathsf{even}}(t), y_\ell^{\mathsf{odd}}(t)\right), \tag{3.18}$$

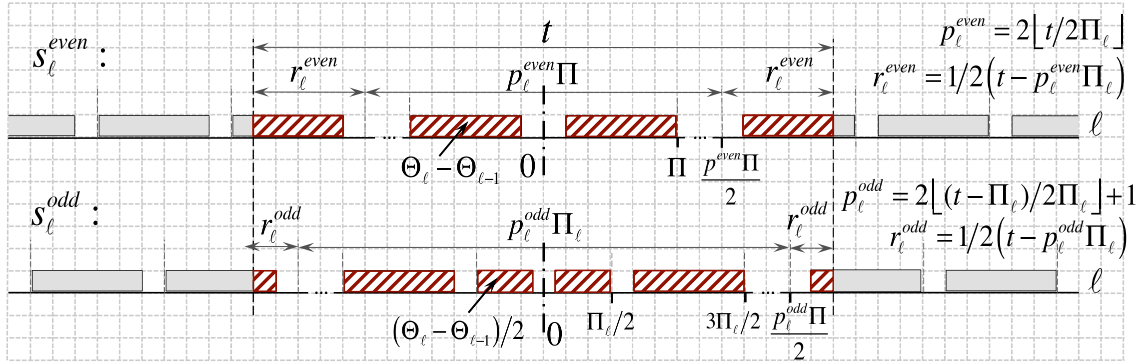


Figure 3.12: The worst-case resource patterns for GMPR with different processor periods

where $y_\ell^{\mathsf{even}}(t)$, $y_\ell^{\mathsf{odd}}(t)$ *originate from the supply patterns* s_ℓ^{even}, s_ℓ^{odd}, *depicted in Fig (put), and computed by*

$$
\begin{aligned}
y_\ell^{\mathsf{even}}(t) &= p_\ell^{\mathsf{even}}(\Theta_\ell - \Theta_{\ell-1}) + 2\,(r_\ell^{\mathsf{even}} - \Pi_\ell + \Theta_\ell - \Theta_{\ell-1})_0 \\
p_\ell^{\mathsf{even}} &= 2\left\lfloor \frac{t}{2\Pi_\ell} \right\rfloor \\
r_\ell^{\mathsf{even}} &= \frac{1}{2}(t - p_\ell^{\mathsf{even}}\Pi_\ell)
\end{aligned}
$$

and

$$
\begin{aligned}
y_\ell^{\mathsf{odd}}(t) &= p_\ell^{\mathsf{odd}}(\Theta_\ell - \Theta_{\ell-1}) + 2\left(r_\ell^{\mathsf{odd}} - \Pi_\ell + \Theta_\ell - \Theta_{\ell-1}\right)_0 \\
p_\ell^{\mathsf{odd}}(t) &= 2\left\lfloor \frac{t - \Pi_\ell}{2\Pi_\ell} \right\rfloor + 1 \\
r_\ell^{\mathsf{odd}} &= \frac{1}{2}\left(t - p_\ell^{\mathsf{odd}}\Pi_\ell\right)
\end{aligned}
$$

Further analysis of GMPR with different processor periods is kept for future work.

3.6 Schedulability over GMPR

The GMPR interface describes the amount of computing resources provided to an application. We can then formulate a schedulability test over the GMPR.

As schedulability test for the application, we choose the extension of the test by Bertogna et al. (2009) to the PSF interface developed by Bini et al. (2009). We choose this condition because it applies to several different application schedulers such as global EDF or global FP, although it assumes constrained deadline tasks, i.e. for all tasks τ_i, $D_i \leq T_i$. While choosing other tests like the one derived in (Baruah et al., 2010) would be possible, the proposed formulation has the advantage of highlighting the constraint on the interface. Thanks to the lossless transformation of a GMPR interface into a PSF (see Section 3.5.2), we can apply directly the schedulability

condition developed over PSF. Below we report, for completeness, the schedulability condition in the simpler expression proposed in (Lipari and Bini, 2010).

Theorem 3 (Theorem 1 in (Lipari and Bini, 2010))**.** *A set of sporadic tasks* $\mathscr{T} = \{\tau_1, \ldots, \tau_n\}$ *with constrained deadlines* $D_i \leq P_i$ *is schedulable on a resource modeled by the PSF functions* $Y_1(t), \ldots, Y_m(t)$*, if*

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k_i=1,\ldots,m} k_i C_i + W_i \leq Y_{k_i}(D_i), \tag{3.19}$$

where W_i *is the maximum* interfering workload *that can be experienced by task* τ_i *in the interval* $[0, D_i]$*, defined as*

$$W_i = \sum_{j=1, j\neq i}^{n} \left(\left\lfloor \frac{D_i}{T_j} \right\rfloor C_j + \min\left\{ C_j, D_i - \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \right\} \right), \tag{3.20}$$

if the application tasks are scheduled by global EDF. Instead if the application tasks are scheduled by global FP

$$W_i = \sum_{j \in \mathsf{hp}(i)} W_{ji}, \tag{3.21}$$

where $\mathsf{hp}(i)$ *denotes the set of indices of tasks with higher priority than i, and* W_{ji} *is the amount of interfering workload caused by* τ_j *on* τ_i*, that is*

$$W_{ji} = N_{ji} C_j + \min\{C_j, D_i + D_j - C_j - N_{ji} T_j\} \tag{3.22}$$

with $N_{ji} = \left\lfloor \frac{D_i + D_j - C_j}{T_j} \right\rfloor$.

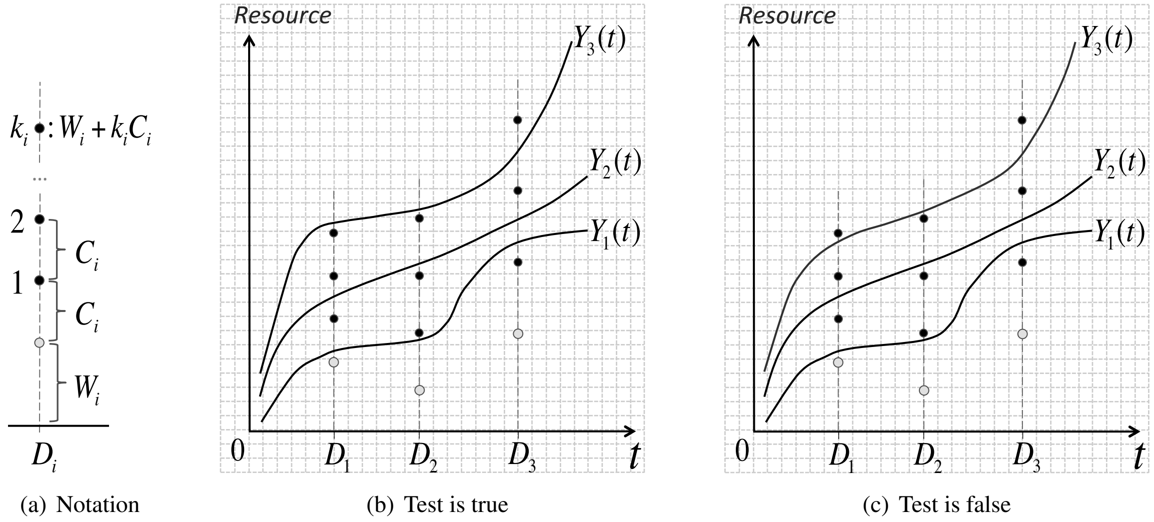
To better understand the schedulability test over PSF of Theorem 3, we illustrate it graphically in Fig. 3.13. In this example we consider a task set $\mathscr{T}$ composed by $n = 3$ tasks. Each task τ_i has an amount of interference W_i, properly determined according to the local scheduling algorithm. For each task τ_i, we draw a dashed vertical line at $t = D_i$. Along this line we represent the quantity W_i denoted as a white dot, and the quantities $W_i + k_i C_i$, with $k_i \in \{1, 2, 3\}$, denoted as black dots. These dots represent the LHS of (3.19). Then we draw the PSF functions $Y_1(t), Y_2(t), Y_3(t)$ as bold continuous lines. In accordance to condition (3.19), task τ_i is schedulable if the k-th dot is not above the Y_k, for some k.

Now consider the case depicted in Fig. 3.13(b). In that case $\mathscr{T}$ is schedulable as the condition (3.19) turns valid for $k_1 \in \{3\}$, $k_2 \in \{2, 3\}$, and $k_3 \in \{1\}$. In Fig. 3.13(c), instead, we show a case when τ_1 cannot be guaranteed by the test of Theorem 3.

Later we exploit such a schedulability condition to compute the GMPR parameters $\Theta_1, \ldots, \Theta_m$ for a given task set.

3.6.1 Simplified schedulability test

The schedulability condition of Theorem 3 has the complexity of $O(nm)$ since it requires to check if for each task $\tau_i \in \mathscr{T}$ exists any value $k_i \in \{1, \ldots, m\}$ satisfying the inequality (3.19). However,



(a) Notation (b) Test is true (c) Test is false

Figure 3.13: Graphical interpretation of the PSF-based schedulability test

we can shrink the set of values of k_i to be tested without making any pessimistic assumption, by exploiting a linear upper bounds of the PSF function.

The PSF function $Y_k(t)$ can be bounded from above by

$$Y_k(t) \leq kt. \tag{3.23}$$

Substituting Eq. (3.23) into the condition (3.19), we get $k_iC_i + W_i \leq k_iD_i$ and thus

$$k_i \geq \frac{W_i}{D_i - C_i}. \tag{3.24}$$

Considering that k_i is integer and by defining $\bar{k}_i$ as

$$\bar{k}_i = \left\lceil \frac{W_i}{D_i - C_i} \right\rceil, \tag{3.25}$$

the schedulability condition (3.19) turns into

$$\bigwedge_{i=1,\dots,n} \bigvee_{k_i=\bar{k}_i,\dots,m} k_iC_i + W_i \leq Y_{k_i}(D_i). \tag{3.26}$$

3.7 Computation of GMPR using pruning

When an application $\mathscr{T} = \{\tau_1, \dots, \tau_n\}$ is given, it is of key importance to select an interface that can guarantee the timing constraints of the application and, at the same time, requires the minimal amount of resource. In (Burmyakov et al., 2012) we proposed an algorithm to generate a GMPR interface for $\mathscr{T}$ assuming integer resource parameters. However, this assumption made

the problem hardly tractable even for a task set with a low utilization. If instead, the interface parameters are assumed continuous, the problem can be attacked and solved more efficiently.

Consider a set of sporadic tasks $\mathscr{T} = \{\tau_1, \ldots, \tau_n\}$ locally scheduled by the global EDF or the global FP scheduler. In this section we describe a method to compute a GMPR interface for $\mathscr{T}$: For a specified period Π and a parallelism m we find the minimal real-valued resources $\Theta_1, \ldots, \Theta_m$ such that $\mathscr{T}$ is schedulable over the GMPR $\langle \Pi, \{\Theta_1, \ldots, \Theta_m\} \rangle$, according to Theorem 3.

Below, in Section 3.7.1 we compute the minimal necessary parallelism for a GMPR for a given application. Then, in Section 3.7.2 we compute the GMPR resource Θ_m, and in Section 3.7.3 we derive a set of techniques to reduce the computation time for Θ_m. Finally, in Sections 3.7.4 and 3.7.5 we generalize our approach by iteratively computing the resources $\Theta_1, \ldots, \Theta_m$ for all levels of parallelism.

3.7.1 Minimal necessary parallelism for GMPR

No valid GMPR interface may exist for an arbitrary small parallelism. Hence, in Theorem 4 we propose a necessary and sufficient condition for the parallelism of a GMPR, assuming Theorem 3 as schedulability test.

Theorem 4. *Consider a set of sporadic tasks $\mathscr{T} = \{\tau_1, \ldots, \tau_n\}$ locally scheduled by the global EDF or the global FP. Then there always exists a feasible GMPR interface for $\mathscr{T}$ with a parallelism $m \geq \max(\bar{k}_1, \ldots, \bar{k}_n)$, with $\bar{k}_i$ as in (3.25). However, no GMPR can satisfy the schedulability condition (3.19) if $m < \max(\bar{k}_1, \ldots, \bar{k}_n)$.*

Proof. To prove the existence of a GMPR with a parallelism at least $m = \max(\bar{k}_1, \ldots, \bar{k}_n)$, we show that $\mu = \langle \Pi, \{\Pi, 2\Pi, \ldots, m\Pi\} \rangle$ is a valid GMPR interface for $\mathscr{T}$. According to Eq. (3.4), the PSF functions for μ are

$$Y_k(t) = kt, \qquad k = 1, \ldots, m.$$

The schedulability condition (3.26) over μ turns into

$$\bigwedge_{i=1,\ldots,n} \bigvee_{k_i=\bar{k}_i,\ldots,m} k_i C_i + W_i \leq k_i D_i.$$

For each τ_i we set $k_i = \bar{k}_i$, and check that the schedulability of $\mathscr{T}$ over μ holds:

$$\left\lceil \frac{W_i}{D_i - C_i} \right\rceil D_i \geq \left\lceil \frac{W_i}{D_i - C_i} \right\rceil C_i + W_i$$
$$\left\lceil \frac{W_i}{D_i - C_i} \right\rceil (D_i - C_i) \geq W_i$$
$$\left\lceil \frac{W_i}{D_i - C_i} \right\rceil \geq \frac{W_i}{D_i - C_i}.$$

Thus, $\mu = \langle \Pi, \{\Pi, 2\Pi, \ldots, m\Pi\} \rangle$ is a valid GMPR for $\mathscr{T}$.

To prove the other direction of the implication, let us denote, without loss of generality, by $\overline{k} = \max(\overline{k}_1, \dots, \overline{k}_n)$ and by ℓ - task index such that $\overline{k} = \overline{k}_\ell$. If $m < \overline{k}$, then the task τ_ℓ can never be guaranteed by (3.26). □ □

According to Theorem 4, we can only compute a GMPR interface for $\mathscr{T}$ with a parallelism $m \geq \max(\overline{k}_1, \dots, \overline{k}_n)$.

3.7.2 Computation of GMPR aggregated resource

When designing an interface of a given application, our primary target is the minimization of the overall resource consumption Θ_m. Before formulating the interface design as an optimization problem, let us denote D_Θ all feasible resources $\Theta_1, \dots, \Theta_m$ satisfying the constraints in Definition (5) of GMPR, so that:

$$(\Theta_1, \dots, \Theta_m) \in D_\Theta \iff \begin{cases} 0 \leq \Theta_{k+1} - \Theta_k \leq \Pi \\ \Theta_{k+1} - \Theta_k \leq \Theta_k - \Theta_{k-1} \\ \Theta_k = 0, \qquad k \leq 0. \end{cases} \tag{3.27}$$

Then we compute Θ_m subject to the schedulability test (3.26):

$$\begin{aligned} &\text{minimize } \Theta_m \\ &\text{subject to} \\ &\quad (\Theta_1, \dots, \Theta_m) \in D_\Theta \\ &\quad \forall i = 1, \dots, n,\ \exists k_i \in \left\{\overline{k}_i, \dots, m\right\} :\ Y_{k_i}(D_i, \Theta_1, \dots, \Theta_{k_1}) \geq k_i C_i + W_i. \end{aligned} \tag{3.28}$$

To solve the optimization problem (3.28), we first have to exclude the $\exists$-quantifiers from it. Therefore, we propose to solve (3.28) for each possible combination $(k_1, \dots, k_n)$, with $k_i \in \{\overline{k}_i, \dots, m\}$, and then to choose the minimal Θ_m over all cases. Below we provide a detailed description of this approach.

Let us denote possible combinations $(k_1, \dots, k_n)$ as $\mathscr{K}_m$ so that

$$\mathscr{K}_m = \left\{(k_1, \dots, k_n) |\ k_i = \overline{k}_i, \dots, m\right\}.$$

For a specific choice of $(k_1, \dots, k_n) \in \mathscr{K}_m$ the optimization problem (3.28) turns into

$$\begin{aligned} &\text{minimize } \Theta_m \\ &\text{subject to} \\ &\quad (\Theta_1, \dots, \Theta_m) \in D_\Theta \\ &\quad \forall i = 1, \dots, n \quad Y_{k_i}(D_i, \Theta_1, \dots, \Theta_{k_i}) \geq k_i C_i + W_i \end{aligned} \tag{3.29}$$

To solve (3.29), we employ the Matlab optimization toolbox. Let us denote the solution of (3.29) as $\Theta_m(k_1,\ldots,k_n)$, if any exists. Then we choose the minimal Θ_m over $\mathscr{K}_m$ as

$$\Theta_m = \min_{(k_1,\ldots,k_n)\in\mathscr{K}_m} \Theta_m(k_1,\ldots,k_n)\,. \tag{3.30}$$

For some combination $(k_1,\ldots,k_n)$ the optimization problem (3.29) may have no feasible solution. However, from Theorem 4, there exists at least one case $(k_1,\ldots,k_n)\in\mathscr{K}_m$ such that (3.28) becomes feasible. Hence, the minimum of (3.30) is well defined.

Next, in Section 3.7.3 we propose a method to reduce the run-time of the optimization problem (3.30) by reducing the search space for the resources $\Theta_1,\ldots,\Theta_m$ and shrinking the enumeration space $\mathscr{K}_m$.

3.7.3 Pruned search space for GMPR resource

To reduce the search space for the GMPR resources $\Theta_1,\ldots,\Theta_m$, we first formulate a set of preliminary constraints in Lemma 2.

Lemma 2. *All feasible GMPR resources* $(\Theta_1,\ldots,\Theta_m)\in D_\Theta$ *defined by (3.27) satisfy the following constraints:*

$$j<k \quad\Rightarrow\quad \Theta_k \le \frac{k}{j}\Theta_j \tag{3.31}$$

Proof. Let us decompose Θ_k as

$$\Theta_k = \sum_{\ell=1}^{k-1}(\Theta_\ell-\Theta_{\ell-1}) + (\Theta_k-\Theta_{k-1})\,. \tag{3.32}$$

From (3.27), each feasible case $(\Theta_1,\ldots,\Theta_m)\in D_\Theta$ satisfies the constraint

$$\Theta_\ell-\Theta_{\ell-1} \ge \Theta_k-\Theta_{k-1} \qquad \ell<k. \tag{3.33}$$

Substituting (3.33) into (3.32) gives us

$$\Theta_k \le \frac{k}{k-1}\Theta_{k-1}. \tag{3.34}$$

Applying mathematical induction to the expression above, we get (3.31):

$$\begin{aligned}\Theta_k &\le \frac{k}{k-1}\Theta_{k-1} \le \frac{k}{k-1}\left(\frac{k-1}{k-2}\Theta_{k-2}\right) \le \\ &\frac{k}{k-1}\,\frac{k-1}{k-2}\cdots\left(\frac{k-i+1}{k-i}\Theta_{k-i}\right) = \frac{k}{k-i}\Theta_{k-i}\end{aligned} \tag{3.35}$$

with $i=1,\ldots,k-1$. □ □

Let $\mathscr{T}$ be a schedulable task set over a GMPR interface $\langle\Pi,\{\Theta_1,\ldots,\Theta_m\}\rangle$ according to condition (3.26). For each task τ_i, let us denote by k_i^* the smallest k_i, in $\{\overline{k}_i,\ldots,m\}$, for which the condition (3.26) is true. Below, we compute a reduced search space for the GMPR resources $\Theta_1,\ldots,\Theta_m$ by exploiting the lower and the upper bounds for $Y_k(t)$ derived in Section 3.5.3:

$$Y_k(t) \;\geq\; \underline{Y}_k(t) = \frac{\Theta_k}{\Pi}t - 2\frac{\Theta_k}{\Pi}\left(\Pi - \frac{\Theta_k}{k}\right) \tag{3.36}$$

$$Y_k(t) \;\leq\; \overline{Y}_k(t) = \frac{\Theta_k}{\Pi}t. \tag{3.37}$$

Consider a task τ_i. The test (3.26) is false for any $k_i < k_i^*$:

$$Y_{k_i}(D_i) < k_i C_i + W_i.$$

Substituting the lower bound (3.36) for $Y_k(t)$ into the condition above, we get the quadratic inequality

$$\left(\frac{2}{k\Pi}\right)\Theta_k{}^2 + \left(\frac{D_i}{\Pi} - 2\right)\Theta_k - (kC_i + W_i) < 0$$

with a solution

$$\begin{aligned} \overline{\Theta}_k{}^* &= \frac{k\Pi}{4}\left(\sqrt{\left(\frac{D_i}{\Pi} - 2\right)^2 + \frac{8}{k\Pi}(kC_i + W_i)} - \left(\frac{D_i}{\Pi} - 2\right)\right), \\ \Theta_k &< \overline{\Theta}_k{}^*. \end{aligned} \tag{3.38}$$

By applying Lemma 2, the constraint above yields the following upper bound for the resource Θ_k denoted as $\overline{\Theta}_k$:

$$\overline{\Theta}_k = \begin{cases} \Pi, & k = k_i^* = 1, \\ \min\left(\overline{\Theta}_k{}^*, \Pi\right), & k < k_i^*,\ k = 1, \\ \min\left(\overline{\Theta}_k{}^*, \dfrac{k}{k-1}\overline{\Theta}_{k-1}\right), & k < k_i^*,\ k \neq 1, \\ \dfrac{k}{k-1}\overline{\Theta}_{k-1}, & k \geq k_i^*,\ k \neq 1. \end{cases} \tag{3.39}$$

with $\overline{\Theta}_k{}^*$ defined by (3.38).

The test (3.26) is true for $k = k_i^*$. Applying the upper bound (3.37) for $Y_k(t)$ in (3.26), we get

$$\Theta_{k_i^*} \geq \frac{\Pi}{D_i}\left(k_i^* C_i + W_i\right), \tag{3.40}$$

that, together with Lemma 2, yields the following lower bound for the resource Θ_k denoted as $\underline{\Theta}_k$:

$$\underline{\Theta}_k = \begin{cases} \frac{k}{k_i^*} \underline{\Theta}_{k_i^*}, & \text{if } k < k_i^*, \\ \frac{\Pi}{D_i} (k_i^* C_i + W_i), & \text{otherwise.} \end{cases} \tag{3.41}$$

Let us denote the search space for task τ_i as $S_\Theta(\tau_i, k_i^*)$ so that

$$S_\Theta(\tau_i, k_i^*) = \left\{ (\Theta_1, \ldots, \Theta_m) \middle| \ \underline{\Theta}_k \le \Theta_k \le \overline{\Theta}_k \right\},$$

where $\underline{\Theta}_k$, $\overline{\Theta}_k$ are computed according to (3.41), (3.39). The resulting search space for a task set $\mathscr{T}$ is then defined as

$$S_\Theta(\mathscr{T}, k_1^*, \ldots, k_n^*) = D_\Theta \bigcap_{i=1,\ldots,n} S_\Theta(\tau_i, k_i^*), \tag{3.42}$$

where D_Θ denotes all feasible GMPR resources $\Theta_1, \ldots, \Theta_m$ satisfying the constraint (3.27).

Consequently, a case $(k_1, \ldots, k_n) \in \mathscr{K}_m$ is feasible if it results in a non-empty search space

$$S_\Theta(\mathscr{T}, k_1, \ldots, k_n) \neq \emptyset, \tag{3.43}$$

otherwise it can be excluded from $\mathscr{K}_m$. According to our experiments, this approach drastically reduces the size of $\mathscr{K}_m$: the reduction is by more than 99,99% in an average case.

3.7.4 Computation of GMPR resources at lower parallelisms

In Section 3.7.2 we computed the GMPR overall resource Θ_m, only. To complete the GMPR specification, we now need to compute the remaining resources $\Theta_{m-1}, \ldots, \Theta_1$, which should be provided at lower concurrencies.

We propose to compute the resource Θ_k recursively, after computing the resources $\Theta_m, \ldots, \Theta_{k+1}$. To do so, we simply update the optimization problem (3.28) by setting the objective function to minimize Θ_k, and by placing the previously found values for $\Theta_m, \ldots, \Theta_{k+1}$ into the optimization constraints.

In this case, rather than repeating the enumeration of $\mathscr{K}_m$ to solve the optimization problem (3.28) for Θ_k, we can further shrink the enumeration space by considering among the feasible cases $(k_1, \ldots, k_n)$ only those ones, which yield the minimal value for Θ_{k+1}. Hence the reduced enumeration space $\mathscr{K}_k$ for Θ_k is given by the equation

$$\begin{aligned} &\mathscr{K}_k \subseteq \mathscr{K}_{k+1} \ : \\ &\quad \forall (k_1, \ldots, k_n) \in \mathscr{K}_{k+1} : \ \Theta_{k+1}(k_1, \ldots, k_n) = \Theta_{k+1}^* \ \to \ (k_1, \ldots, k_n) \in \mathscr{K}_k, \end{aligned} \tag{3.44}$$

where Θ_{k+1}^* denotes the found minimal value for Θ_{k+1}.

The computation time for Θ_k is significantly lower compared to Θ_{k+1}, what is due to a shrunk enumeration space $\mathscr{K}_k$, and a lower number of optimization variables.

3.7.5 Algorithm to compute GMPR

Finally, we conclude by proposing an algorithm that assigns the minimal GMPR resources $\Theta_1, \ldots, \Theta_m$ such that a given task set $\mathscr{T}$ is schedulable over an interface. As a schedulability condition, we choose the one in (3.26). We recall that the period Π and the parallelism m for a searching GMPR are given.

Step 1: For each task τ_i compute $\bar{k}_i$ as defined in (3.25).

Step 2: Check whether the necessary condition for m (Theorem 4) is met:

$$m \geq \max(\bar{k}_1, \ldots, \bar{k}_n).$$

If the condition above is violated, report the nonexistence of a valid GMPR interface for $\mathscr{T}$ with a specified m, and terminate the algorithm.

Step 3: Generate the enumeration space $\mathscr{K}_m$ such that

$$\mathscr{K}_m = \{(k_1, \ldots, k_n) |\ k_i = \bar{k}_i, \ldots, m\}$$

satisfying the condition (3.43).

Step 4: Compute Θ_m: for each case $(k_1, \ldots, k_n) \in \mathscr{K}_m$ determine the search space according to (3.42), solve the optimization problem (3.28), and then choose the minimal Θ_m over $\mathscr{K}_m$.

Step 5: Compute Θ_k recursively after computing $\Theta_m, \ldots, \Theta_{k+1}$:

(a) Define $\mathscr{K}_k$ from Eq. (3.44) so that any $(k_1, \ldots, k_n) \in \mathscr{K}_{k+1}$ resulted in the optimal Θ_{k+1} is included into $\mathscr{K}_k$.

(b) Substitute the computed values for $\Theta_m, \ldots, \Theta_{k+1}$ into the optimization constraints of (3.28), and minimize Θ_k subject to these constraints. Solve the resulting optimization problem over $\mathscr{K}_k$, and then choose the minimal Θ_k.

Step 6: Follow the Step 5 to compute all the resources $\Theta_{m-1}, \ldots, \Theta_1$. In the end, $\langle \Pi, \{\Theta_1, \ldots, \Theta_m\} \rangle$ is the sought-for interface for $\mathscr{T}$ having the minimized resources $\Theta_1, \ldots, \Theta_m$.

Algorithm complexity. The complexity of the algorithm to compute a GMPR interface depends on the complexity of the optimization problem (3.29). Due to the presence of the PSF function $Y_k(t)$, which is non-convex, the optimization problem (3.29) is non-convex. Although the complexity of such problems remains to be an open problem in the literature, it is generally considered as exponential, until the opposite is proved (Ausiello et al., 2008). Thus, the resulting complexity of the proposed algorithm is exponential.

Customized computation of GMPR. We proposed an algorithm to compute a GMPR interface having the minimized resources $\Theta_1, \ldots, \Theta_m$. At the same time, our approach is easily extendable

for computing a customized GMPR interface, which meets specific user requirements (e.g. a constraint on the maximum resource fraction to be provided at each concurrency), rather than simply having the minimized consumed resources. In this case the custom constraints should be incorporated in the optimization problem (3.29).

3.8 Scheduling GMPR interfaces

Once the resource demand of each component is abstracted by an interface, these interfaces should be scheduled upon a hardware platform. To schedule GMPR interfaces, we now introduce a notion of interface tasks. A set of *interface* tasks for a GMPR interface $\langle \Pi, \{\Theta_1, \ldots, \Theta_m\} \rangle$ is comprised of m implicit-deadline $(D = T)$ periodic tasks such that:

$$\mathscr{T}' = \{\tau_1' = (C_1', \Pi), \ldots, \tau_m = (C_m', \Pi)\}, \tag{3.45}$$

where the execution time equals to

$$C_k' = (\Theta_k - \Theta_{k-1}).$$

(We set $\Theta_0 = 0$ for convenience.)

The interface tasks in $\mathscr{T}'$ have an identical period T equal to the period of a GMPR interface Π. Clearly, the overall resource demand of $\mathscr{T}'$ over a period Π is $\sum_{k=1}^{m} C_k' = \Theta_m$.

To schedule GMPR interfaces, we first transform each one into interface tasks following (3.45), and then we employ any suitable policy to schedule the resulting periodic tasks.

The notion of interface tasks supports another important property for hierarchical systems, which is called *composability:* by the given GMPR interfaces of child components we can compute a GMPR interface of a parent component.

3.9 Evaluation

In this section, we compare the amount of resource used by GMPR and MPR to feasibly schedule randomly generated task sets. For each experiment setting, we compute the minimal GMPR and MPR interfaces by employing the algorithm described in Section 3.7.5.

The algorithm to compute interfaces and the scenarios of the experiments have been implemented in Matlab, and they are publicly available at `https://sites.google.com/site/artemburmyakov/home/papers`.

3.9.1 Task set generation

Synthetic task sets $\mathscr{T} = \{\tau_i = (C_i, T_i)\}$ are randomly generated by specifying the total task set utilization $U_{\mathscr{T}}$, the maximum individual task utilization $U_{\max}$, and the ratio between the maximum and the minimum periods $T_{\max}/T_{\min}$. In our random generation method, the number of tasks in $\mathscr{T}$

Table 3.3: Key parameters: default values

Parameter	Default value
Task set utilization, $U_{\mathscr{T}}$	2.5
Maximum individual task utilization, $U_{\max}$	0.3
Minimum task period, $T_{\min}$	20
Ratio between the maximum and the minimum task periods, $T_{\max}/T_{\min}$	10
Interface period, Π	20
Parallelism increment, Δm	3

is not fixed. Instead, it is implicitly determined as the total utilization of $\mathscr{T}$ reaches the specified value $U_{\mathscr{T}}$.

The minimum period $T_{\min}$ is set to 20 and all task periods are randomly generated so that the specified ratio $T_{\max}/T_{\min}$ is not violated.

3.9.2 Experiments: Resource gain

We evaluate the resource gain of GMPR over MPR for the parameters listed in Table 3.3. In each experiment, we compare the interfaces utilization as one parameter varies, while the rest are left equal to the default values reported in Table 3.3.

In each experiment, we randomly generate at least 200 task sets, and then we plot the average interface utilizations $\frac{\Theta_m}{\Pi}$ among these task sets, as well as the relative GMPR gain.

For each generated task set, the interface parallelism is set to

$$m = m_{\min} + \Delta m,$$

where $m_{\min}$ is the minimal parallelism defined by Theorem 4, and the increment Δm is varied through the experiments.

The gain of GMPR over MPR is computed as

$$\mathsf{gain}_{\mathsf{GMPR}} = \frac{U_{\mathsf{MPR}} - U_{\mathsf{GMPR}}}{U_{\mathsf{GMPR}}},$$

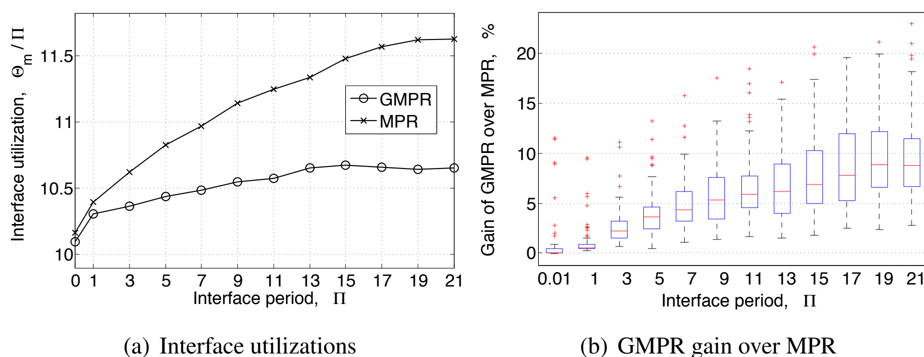
where U_{MPR} denotes the MPR utilization $\frac{\Theta}{\Pi}$, and U_{GMPR} is the GMPR utilization $\frac{\Theta_m}{\Pi}$.

Interface period Π

First, we analyze the GMPR gain for a varying interface period Π. The resulting utilizations of both GMPR and MPR interfaces are plotted in Fig. 3.14(a). For such settings the average GMPR gain is in the order of 5–10%, and it increases for the increasing Π.

The observed trend for gain increase is justified by an expanding search space for the GMPR resources together with Π, which results in a higher degree of freedom for GMPR over MPR.

In Fig. 3.14(b) we also illustrate the gain variability using a boxplot diagram (McGill et al., 1979). In this diagram, the central horizontal mark on each box is the median for the observed gain, the horizontal edges of the box are the 25th and the 75th percentiles, the dashed lines extend to the most extreme gains covering 99.3% observed cases, and the outliers are depicted individually as crosses.



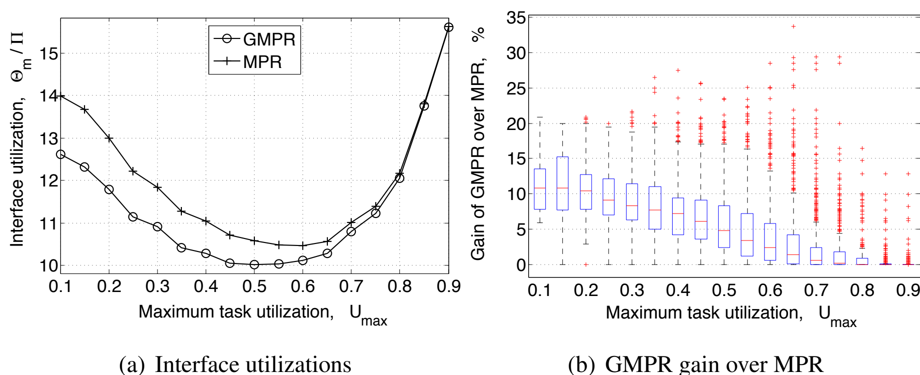
(a) Interface utilizations (b) GMPR gain over MPR

Figure 3.14: Evaluation: GMPR gain for interface period Π

Maximum task utilization U_{max}

In the next experiment, we explore the dependency of the interface utilization on the weight of individual tasks, by varying the maximum task utilization U_{max}. The results are reported in Fig. 3.15. The interface utilization is minimal for U_{max} closer to 0.5–0.6, and it drastically increases for U_{max} tending to 0 or 1. We believe that this behavior is influenced by our choice of schedulability test (Lipari and Bini, 2010) used to compute interfaces.

The GMPR gain itself is maximized for lower U_{max}, reaching up to 10–15%, and the gain vanishes as U_{max} tends to 1.

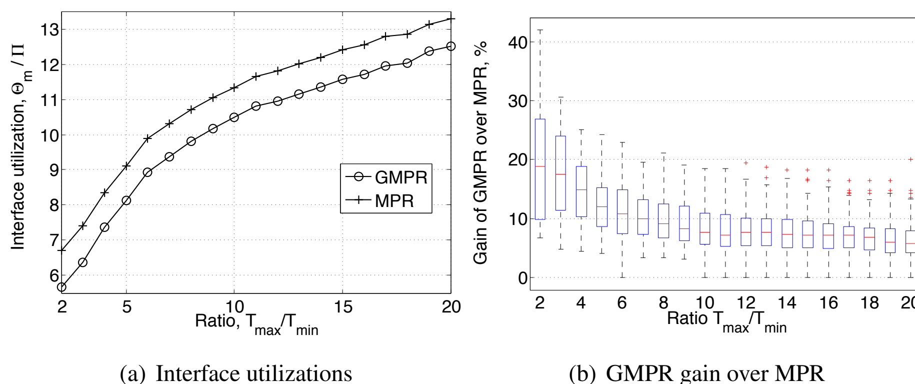


(a) Interface utilizations (b) GMPR gain over MPR

Figure 3.15: Evaluation: GMPR gain for maximum task utilization U_{max}

Task periods ratio $\frac{T_{max}}{T_{min}}$

In Fig. 3.16 we provide the experimental results for a varying ratio T_{max}/T_{min}. The interface utilization significantly increases together with the ratio T_{max}/T_{min}, but the GMPR gain is maximized for lower T_{max}/T_{min}, reaching up to 15–25%.



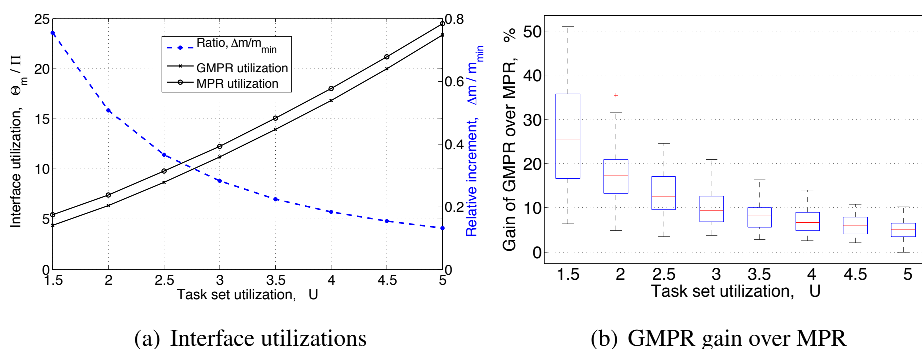
(a) Interface utilizations (b) GMPR gain over MPR

Figure 3.16: Evaluation: GMPR gain for ratio of task periods T_{max}/T_{min}

The observed utilization increase for both GMPR and MPR interfaces with respect to T_{max} is justified by the nature of the chosen schedulability test, described in Theorem 3. In fact, for fixed parameters U and U_{max}, increasing task periods result in a higher interference of jobs accross the deadline window (so called "carry-in", defined by equation (3.20)), increasing the overall utilization of an interface.

Task set utilization $U_{\mathcal{T}}$

We also analyze the gain of GMPR over MPR as the task set utilization $U_{\mathcal{T}}$ varies. The results are depicted in Fig. 3.17. In this case the gain decreases for increasing $U_{\mathcal{T}}$. A reason for such behavior is that, although the absolute parallelism increment Δm remains constant, its relative proportion $\Delta m/m_{min}$ decreases (see Fig. 3.17(a)), due to m_{min} increasing with $U_{\mathcal{T}}$, resulting in a reduced scope for parallelism.

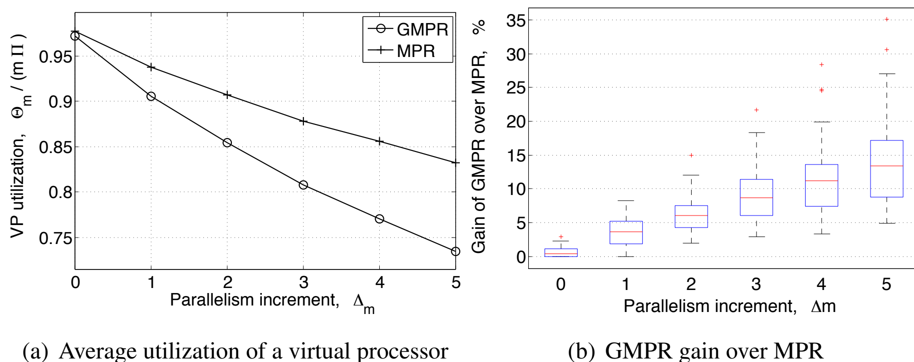


(a) Interface utilizations (b) GMPR gain over MPR

Figure 3.17: Evaluation: GMPR gain for task set utilization $U_{\mathcal{T}}$

Parallelism increment Δm

In the last experiment we analyze the relation between an average utilization of a virtual processor, $\frac{\Theta_m}{m\Pi}$, and the parallelism increment Δm. The results are provided in Fig. 3.18. As expected, an average utilization of a virtual processor reduces for increasing parallelism of an interface.



(a) Average utilization of a virtual processor (b) GMPR gain over MPR

Figure 3.18: Evaluation: GMPR gain for the parallelism increment Δm

The GMPR gain itself increases together with Δm. Such a dependency is expectable since an increased Δm leads to a higher degree of freedom for GMPR over MPR, allowing a larger margin to minimize the consumed resource.

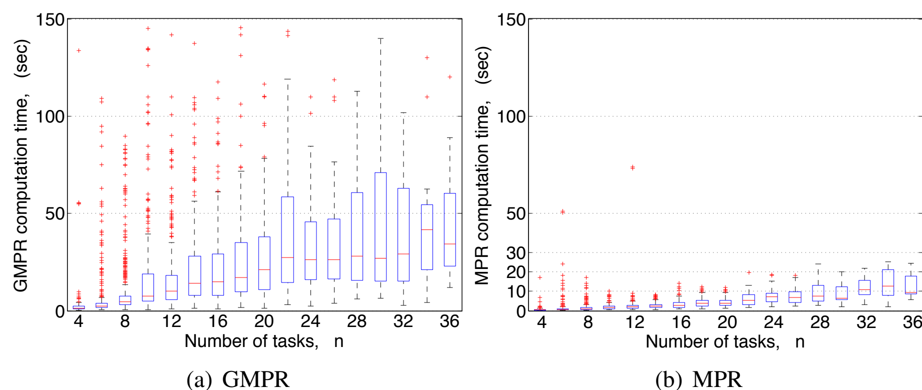
We also notice that the utilization of both GMPR and MPR is minimal for $\Delta m = 0$, and it increases with Δm. This observation confirms the result of Shin et al. (2008) regarding the minimum utilization of a multiprocessor interface, and moreover, this result looks to be independent of the schedulability test used to compute an interface.

3.9.3 Experiments: Runtime analysis

In this experiment we analyze a set of performance metrics for the algorithm to compute a GMPR interface, based on the solution of the resource minimization problem, as described in Section 3.7. The algorithm has been implemented in the Matlab 2010 environment. The experiment has been performed on a hardware platform with the following specifications:

- Processor: Intel(R) Core(TM) i7-3630QM CPU @ 2.40 GHz
- Operating memory (RAM): 8,00 GB
- System type: 64-bit

In Tables 3.4, 3.5 and in Fig. 3.19 we report the measured run-time for the GMPR computation, for a varying number of tasks n and the parallelism m. Although the proposed algorithm to compute GMPR is considered to have an exponential complexity, the results show a linear increase of the algorithm run-time over n and m. This result confirms the effectiveness of the search space reduction mechanism derived in Section 3.7.3.



(a) GMPR (b) MPR

Figure 3.19: Evaluation: comparison of computation time for GMPR and MPR. The percentiles are 25% and 75%.

The computation time for MPR is 2–5 times lower compared to GMPR, what is due to a simpler PSF function in the optimization constraints of (3.29).

Table 3.4: The performance metrics for $m = 5$.

n	GMPR time, (sec)	MPR time, (sec)	size of $\mathscr{K}_m$	size reduction, (times)
1–10	< 10	< 1	1–25	1–50
11–18	1–20	1–10	10–50	10^2–10^4
19–25	10–50	1–15	50–120	10^3–10^6
26–30	25–100	5–25	100–200	10^4–10^7
31–35	50–150	10–50	100–300	10^5–10^{10}

Table 3.5: The performance metrics for $m = 10$.

n	GMPR time, (sec)	MPR time, (sec)	size of $\mathscr{K}_m$	size reduction, (times)
1–10	< 10	< 1	5–50	10–1000
11–18	5–100	1–10	10–100	10^2–10^6
19–25	50–250	10–50	50-150	10^5–10^9
26–30	100–300	20-100	< 400	10^8–10^{13}
31–35	100–500	25-150	< 400	10^8–10^{16}

In addition, we have evaluated the performance of several optimization solvers available in the Matlab, as they significantly affect the overall run-time of the GMPR computation. Although the interior-point algorithm finds a more precise solution for (3.29), we have chosen the active-set algorithm for its 5-100 times faster performance, and its acceptable error which is at most 0.05%, and the failure ratio of at most 2% (in case the active-set fails, we employ the interior-point instead).

In Tables 3.4, 3.5 we report the size of the reduced search space $\mathscr{K}_m$ defined by Eq. (3.43). In each case, this value corresponds to the number of optimization problems (3.29) to be resolved in order to determine the minimal GMPR interface. To analyze the efficiency of the search space reduction algorithm, proposed in Section 3.7.3, we also indicate the relative size reduction of $\mathscr{K}_m$ compared to the original search space defined by the schedulability test (3.26). We observe an exponential size reduction of $\mathscr{K}_m$ over a number of tasks n and an interface parallelism m.

3.10 Summary

Motivated by the need to save resource, we introduced the Generalized Multiprocessor Periodic Resource model (GMPR), as an interface of a multiprocessor virtual platform, and proposed a schedulability test for a set of sporadic tasks over GMPR.

Since GMPR is a generalization of the previously proposed Multiprocessor Periodic Resource model (MPR, by Shin et al. (2008)), it can consume at most as much as MPR. Our evaluation confirmed that the resource gain of GMPR over MPR increases together with the period and the parallelism of an interface. The GMPR gain is especially noticable for task sets with smaller individual tasks' utilizations and a shorter range of tasks' periods.

We also addressed the problem of computing a GMPR interface for a given set of sporadic tasks, objecting to minimize the overall amount of resource required by an interface. This problem was modeled as an optimization problem, which turned to be efficiently solvable thanks to the derived tight lower and upper bounds for the solution search space. Such an approach is easily extendable to compute a customized GMPR interface, which meets specific user requirements rather than simply has the minimized consumed resource.

For the future, our primary objective is to explore the flexibility of the GMPR model in deriving a tighter schedulability analysis, specifically dedicated for it. We also consider extending GMPR to a case of asynchronous virtual processors with different periods.

Chapter 4

Conclusion

In this work we derived a set of methods for the schedulability analysis of multiprocessor real-time systems, by exploring the idea of a search space pruning. We have shown that, despite of the NP-hardness of the considered problems, the pruning of a search space allows to obtain solutions in a significantly lower computation time, compared to the state-of-the-art approaches. In the best case, such an approach can reduce the exponential runtime complexity to polynomial or even linear.

An efficient pruning method should provide a good balance between its computation cost and the achieved reduction of the search space. A tight pruning constraint of a high computation cost does not necessarily outperforms a less accurate pruning constraint with a lower computation cost.

All solutions derived in this work significantly outperform the other state-of-the-art approaches, in terms of computation time and space complexity. These solutions have been implemented in C++ and Matlab environments, and are publicly available. Our contributions are listed in details in Section 1.7.

For a future work, we keep the following open problems.

Although an exact schedulability test, derived in Section 2.5, significantly outperforms the other state-of-the-art tests, it remains intractable for more realistic systems with a larger number of tasks. Thus, we are searching for more efficient pruning constraints, to reduce the computation time of the test.

Another open problem is an efficient implementation of an exact schedulability test for event-driven scheduling, proposed in Section 2.6. The main difficulty is the condition for matching system states (see Eq. (2.47) in Section 2.6). Without such a condition implemented, the space complexity of the test jumps from polynomial to exponential, what also results in a significant increase of runtime. We are currently working on this problem.

For compositional multiprocessor scheduling, addressed in Section 3, it remains unclear how to choose an interface period Π for the periodic resource models. Another problem is an efficient computation of the GMPR interface with different periods for virtual processors, introduced in Section 3.5.4.

Appendix A

Proofs

A.1 Proof of Theorem 1

The proof of Theorem 1 relies on the following lemma.

Lemma 3. *Let R, R' be release sequences, as defined in Theorem 1. Let S, S' be resource schedules for R, R' respectively, defined by (2.3).*

S and S' have the property:

$$\forall i,t: \; q'_i(t) = 1 \quad \longrightarrow \quad s_i(t) = s'_i(t), \tag{A.1}$$

where $q'_i(t)$ is defined by (2.2), and $s_i(t)$, $s'_i(t)$ are defined by (2.3), for S and S' respectively.

Observe that Lemma 3 does not require feasibility of $\mathscr{T}$.

To proof Lemma 3, we explore definition (2.3) of supply function $s_i(t)$: τ_i job gets supply over time $[t, t+1)$ (that is $q_i(t) = 1 \wedge s_i(t) = 1$), iff the number of higher priority jobs at t is less than m (that is $\sum_{\ell=1}^{i-1} q_\ell(t) < m$):

$$s_i(t) = 1 \quad \Leftrightarrow \quad \begin{cases} q_i(t) = 1 \\ \sum\limits_{\ell=1}^{i-1} q_\ell(t) < m \end{cases}, \tag{A.2}$$

$$s_i(t) = 0 \quad \Leftrightarrow \quad \left[\begin{array}{l} q_i(t) = 0 \\ \begin{cases} q_i(t) = 1 \\ \sum\limits_{\ell=1}^{i-1} q_\ell(t) \geq m \end{cases} \end{array}\right., \tag{A.3}$$

$$\forall i,\ t$$

where square brace, "[", denotes logical OR.

Also, the necessary constraints for values of $s_i(t)$ are:

$$s_i(t) = 1 \quad \Rightarrow \quad \sum_{\ell=1}^{i-1} q_\ell(t) < m, \tag{A.4}$$

$$s_i(t) = 0 \wedge q_i(t) = 1 \quad \Rightarrow \quad \sum_{\ell=1}^{i-1} q_\ell(t) \geq m, \tag{A.5}$$

$$\forall i,\ t$$

We are now ready to prove the lemma.

Proof. We prove the lemma by contradiction. Suppose that (A.1) is violated for some task τ_{i^*} at time t^*, that is

$$\exists i^*, t^* : \quad q'_{i^*}(t^*) = 1 \wedge s_{i^*}(t^*) \neq s'_{i^*}(t^*). \tag{A.6}$$

Without loss of generality, let t^* be the earliest time, when (A.1) is violated, meaning that

$$\forall i,\ t < t^* : q'_i(t) = 1 \quad \rightarrow \quad s_i(t) = s'_i(t). \tag{A.7}$$

As $s_i(t)$, $s'_i(t)$ are boolean, (A.6) might hold in two cases:

$$\begin{cases} s_{i^*}(t^*) = 1 \\ s'_{i^*}(t^*) = 0 \\ q'_{i^*}(t^*) = 1 \end{cases} \quad \text{(A.8)} \qquad \text{or} \qquad \begin{cases} s_{i^*}(t^*) = 0 \\ s'_{i^*}(t^*) = 1 \\ q'_{i^*}(t^*) = 1 \end{cases} \quad \text{(A.9)}$$

We next show that both cases are infeasible, meaning that (A.6) cannot hold.

Case 1: Substituting (A.4), (A.5) into (A.8), we get that

$$\begin{cases} s_{i^*}(t^*) = 1 \\ s'_{i^*}(t^*) = 0 \\ q'_{i^*}(t^*) = 1 \end{cases} \overset{(A.4)}{\Rightarrow} \begin{cases} \sum_{\ell=1}^{i^*-1} q_\ell(t^*) < m \\ s'_{i^*}(t^*) = 0 \\ q'_{i^*}(t^*) = 1 \end{cases} \overset{(A.5)}{\Rightarrow} \begin{cases} \sum_{\ell=1}^{i^*-1} q_\ell(t^*) < m \\ \sum_{\ell=1}^{i^*-1} q'_\ell(t^*) \geq m \end{cases},$$

meaning that exists such a task $\tau_{i^{**}}$, with $i^{**} < i^*$, that

$$q'_{i^{**}}(t^*) = 1 \tag{A.10}$$

$$q_{i^{**}}(t^*) = 0. \tag{A.11}$$

From (A.10), $\tau_{i^{**}}$ has a job at time t^*, in S'. Let J'_{**} denote that job, and let $t'_{r^{**}}$ denote its release time, such that

$$r'_{i^{**}}(t'_{r^{**}}) = 1. \tag{A.12}$$

As J'_{**} has not been completed by time t^*, J'_{**} has not received C_i units of resource by time t^*:

$$\sum_{t=t'_{r^{**}}}^{t^*-1} s'_{i^{**}}(t) < C_{i^*{*}}. \tag{A.13}$$

From (2.12), whenever $\tau_{i^{**}}$ releases a job in R', it also releases a job in R. Due to (A.12) and (2.12),

$$r'_{i^{**}}(t'_{r^{**}}) = 1 \quad \overset{(2.12)}{\Longrightarrow} \quad r_{i^{**}}(t'_{r^{**}}) = 1,$$

that is $\tau_{i^{**}}$ releases a job at time t'_{r**}, in R. Let J_{**} denote that job, as well as $t_{c^{**}}$ - the completion time for J_{**}.

Considering (A.11), τ_i has no job pending at time $t^* \geq t'_{r^{**}}$ in S, meaning that J_{**} has been completed by time t^*. Thus, it follows that:

$$t_{c^{**}} \leq t^*$$
$$\sum_{t=t'_{r^{**}}}^{t_{c^{**}}-1} s_{i^{**}}(t) = C_i,$$

what together with (A.13) yields contradiction to (A.7), as

$$\sum_{t=t'_{r^{**}}}^{t_{c^{**}}-1} s_{i^{**}}(t) = C_i \quad \bigwedge \quad \sum_{t=t'_{r^{**}}}^{t_{c^{**}}-1} s'_{i^{**}}(t) < C_i$$

implies that

$$\exists t \in [t'_{r^{**}}, t^*): \; q'_{i^{**}}(t) = 1 \wedge s_{i^{**}}(t) \neq s'_{i^{**}}(t) \tag{A.14}$$

Thus, Eq. (A.8) is infeasible.

Case 2: Suppose that (A.9) holds. Substituting (A.2), (A.3) into (A.9), we get that

$$\begin{cases} s_{i^*}(t^*) = 0 \\ s'_{i^*}(t^*) = 1 \\ q'_{i^*}(t^*) = 1 \end{cases} \overset{(A.3)}{\Longleftrightarrow} \begin{cases} \left[\begin{array}{l} q_{i^*}(t^*) = 0 \\ \begin{cases} q_{i^*}(t^*) = 1 \\ \sum_{\ell=1}^{i^*-1} q_\ell(t^*) \geq m \end{cases} \end{array}\right. \\ s'_{i^*}(t^*) = 1 \\ q'_{i^*}(t^*) = 1 \end{cases} \overset{(A.2)}{\Longleftrightarrow}$$

$$\begin{cases} \left[\begin{array}{l} q_{i^*}(t^*) = 0 \\ \begin{cases} q_{i^*}(t^*) = 1 \\ \sum_{\ell=1}^{i^*-1} q_\ell(t^*) \geq m \end{cases} \end{array}\right. \\ q'_{i^*}(t^*) = 1 \\ \sum_{\ell=1}^{i^*-1} q'_\ell(t^*) < m \end{cases} \Longrightarrow \left[\begin{array}{l} \begin{cases} q_{i^*}(t^*) = 0 \\ q'_{i^*}(t^*) = 1 \end{cases} \\ \begin{cases} q_{i^*}(t^*) = 1 \\ \sum_{\ell=1}^{i^*-1} q_\ell(t^*) \geq m \\ \sum_{\ell=1}^{i^*-1} q'_\ell(t^*) < m \end{cases} \end{array}\right.$$

The first case

$$q_{i^*}(t^*) = 0$$
$$q'_{i^*}(t^*) = 1$$

is infeasible; the proof is conducted by analogy to Case 1, for Eq. (A.10), (A.11).

Suppose that the second case holds:

$$q_{i^*}(t^*) = 1 \tag{A.15}$$
$$\sum_{\ell=1}^{i^*-1} q_\ell(t^*) \geq m \tag{A.16}$$
$$\sum_{\ell=1}^{i^*-1} q'_\ell(t^*) < m \tag{A.17}$$

Due to (A.16) and (A.17), there exists such a task $\tau_{i^{**}}$, with $i^{**} < i^*$, that satisfies the constraints:

$$q_{i^{**}}(t^*) = 1 \tag{A.18}$$
$$\sum_{\ell=1}^{i^{**}} q_\ell(t^*) = m \tag{A.19}$$
$$q'_{i^{**}}(t^*) = 0. \tag{A.20}$$

Let J_{**} denote $\tau_{i^{**}}$ job, pending at time t^* in S (there is such a job, due to (A.18)), and let $t_{r^{**}}$ denote

its release time:

$$r_{i^{**}}(t_{r^{**}}) = 1,$$
$$\sum_{t=t_{r^{**}}}^{t^*} s_{i^{**}}(t) < C_i. \tag{A.21}$$

Observe that J_{**} interfers with a lower priority job at time t^*: a processor is assigned to J_{**} at time t^* (that is due to (A.18) and (A.19)), and the number of pending jobs at t^* exceeds m (that is due to (A.15) and (A.16)). From definition (2.12) of R':

$$\begin{cases} r_{i^{**}}(t_{r^{**}}) = 1 \\ \text{(2.11) holds for } J_{**} \end{cases} \overset{(2.12)}{\Longrightarrow} r'_{i^{**}}(t_{r^{**}}) = 1,$$

meaning that $\tau_{i^{**}}$ released a job at time $t_{r^{**}}$, in R'. Let J'_{**} denote that job, and $t'_{c^{**}}$ - its completion time. Due to (A.20), $\tau_{i^{**}}$ has no pending job at t^*, in S', meaning that J'_{**} has been completed by time t^*:

$$t'_{c^{**}} \leq t^*$$
$$\sum_{t=t_{r^{**}}}^{t'_{c^{**}}-1} s'_{i^{**}}(t) = C_i$$

what together with (A.21) contradicts to assumption (A.7):

$$\begin{cases} t'_{c^{**}} \leq t^* \\ \sum_{t=t_{r^{**}}}^{t'_{c^{**}}-1} s'_{i^{**}}(t) = C_i \end{cases} \bigwedge \sum_{t=t_{r^{**}}}^{t^*} s_{i^{**}}(t) < C_i$$

$$\sum_{t=t'_{r^{**}}}^{t'_{c^{**}}-1} s'_{i^{**}}(t) = C_i \bigwedge \sum_{t=t'_{r^{**}}}^{t'_{c^{**}}-1} s_{i^{**}}(t) < C_i,$$

meaning that (A.7) is violated:

$$\exists t \in [t_{r^{**}}, t^*) : q_{i^{**}}(t) = 1 \wedge s_{i^{**}}(t) \neq s'_{i^{**}}(t)$$

We conclude that both cases, (A.8) and (A.9), are infeasible, and lemma statement (A.1) holds always.

□

We finally prove Theorem 1.

Proof for Theorem 1. Necessity: We first prove that feasibility of R implies feasibility of R'. The proof is conducted by contradiction. Suppose that exist such R, R', satisfying (2.12), that R is feasible, but R' is infeasible. For R', let t_{dm} denote the earliest missed deadline for τ_k; consequently,

$t_{\mathsf{dm}} - D_k$ is the release time of the job missing the deadline:

$$r'_k(t_{\mathsf{dm}} - D_k) = 1, \tag{A.22}$$

$$\sum_{t=t_{\mathsf{dm}}-D_k}^{t_{\mathsf{dm}}-1} s'_k(t) < C_k. \tag{A.23}$$

Due to (2.12), if τ_k has released a job at time $(t_{\mathsf{dm}} - D_k)$ in R', then it has also released a job in R:

$$r_k(t_{\mathsf{dm}} - D_k) = 1. \tag{A.24}$$

Due to the assumption that R is feasible,

$$\sum_{t=t_{\mathsf{dm}}-D_k}^{t_{\mathsf{dm}}-1} s_k(t) = C_k, \tag{A.25}$$

what contradicts to (A.23) due to Lemma 3.

Sufficiency: The sufficiency proof shows that feasibility of R' implies feasibility of R. Such a proof is conducted by analogy to the necessity proof, by swapping functions $r_k(t)$, $s_k(t)$ and $r'_k(t)$, $s'_k(t)$ in Eq. (A.22)-(A.25).

The theorem follows.

□

A.2 Proof of Corollary 2

Let τ_k denote a task under the schedulability analysis. Let us assume that tasks $\tau_1, \ldots, \tau_{k-1}$ are schedulable. The length of the longest schedule, satisfying Corollary 1, is less than $\bar{t}$, computed by any of the following equations:

$$\bar{t} = \sum_{i=1}^{k} D_i \tag{A.26}$$

$$\bar{t} = \max(C_1, \ldots, C_m) + \sum_{i=m+1}^{k} D_i \tag{A.27}$$

$$\bar{t} = \max(C_1, \ldots, C_m) + \sum_{i=m+1}^{k-1} R_i + D_k, \tag{A.28}$$

where R_i denotes the worst-case response time for task τ_i.

Eq. (A.26) and (A.27) originate from the release scenarios depicted in Figs. A.1, A.2. (The figures assume that time is discrete.)

Below we prove each Eq. (A.26)-(A.28).

Let us first introduce a set of auxiliary definitions. Consider an arbitrary release scenario R for tasks $\tau_1, \ldots, \tau_k$, satisfying Corollary 1. For such R, let us define a sequence of interfering jobs $I = \{j_{i_1}, \ldots, j_{i_{k^*}}\}$ as follows:

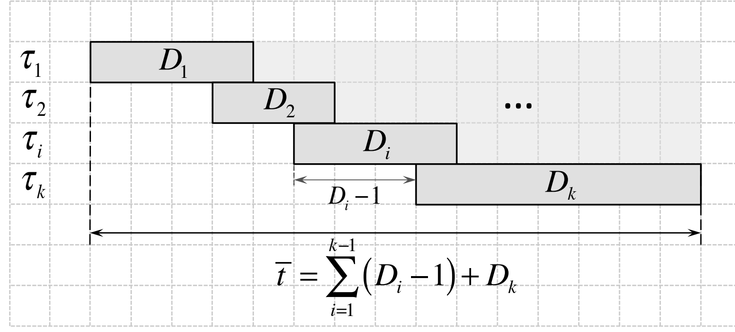


Figure A.1: An upper bound $\bar{t}$ on the length of the longest schedule, satisfying Corollary 1. Such a bound $\bar{t}$ does not depend on the number of available processors. The figure assumes discrete time.

(a) Job j_{i_1} is released by task τ_{i_1} at time $t = 0$. If several jobs are released at time $t = 0$ simultaneously, then j_{i_1} has the highest priority among them.

(b) Job j_{i_ℓ} of task τ_{i_ℓ} interferes with a lower priority job $j_{i_{\ell+1}}$ of task $\tau_{i_{\ell+1}}$ (with $i_{\ell+1} > i_\ell$), for $\ell = 1, \ldots, k^* - 1$. (Due to Corollary 1, for any job j_{i_ℓ}, there should exist such a job $j_{i_{\ell+1}}$.) If job j_{i_ℓ} interferes at different times with different jobs, then $j_{i_{\ell+1}}$ is the first interfered job, and if job j_{i_ℓ} interferes with different jobs simultaneously, then $j_{i_{\ell+1}}$ has the highest priority among them.

(c) Job $j_{i_{k^*}}$ is released by task τ_k, that is $i_{k^*} = k$.

To clarify the definition of I above, we provide an example. In Fig. A.3, we depict several release scenarios for tasks τ_1, τ_2, τ_3, and the respective values for I in each case.

We recall that, when analyzing the schedulability of task τ_k, it is sufficient to consider only those release sequences, wherein task τ_k releases not more than one job. The details are provided in Corollary 1.

For each job j_{i_ℓ} in the definition of I above, let $t^{\mathsf{r}}_{i_\ell}$ and $t^{\mathsf{c}}_{i_\ell}$ denote the respective release and completion times for a job j_{i_ℓ}. The following constraints should hold for $t^{\mathsf{r}}_{i_\ell}$ and $t^{\mathsf{c}}_{i_\ell}$:

$$t^{\mathsf{r}}_{i_1} = 0 \tag{A.29}$$

$$t^{\mathsf{r}}_{i_{\ell+1}} < t^{\mathsf{c}}_{i_\ell} \tag{A.30}$$

$$t^{\mathsf{c}}_{i_\ell} \leq t^{\mathsf{r}}_{i_\ell} + D_{i_\ell}, \tag{A.31}$$

$$\ell = 1, \ldots, k^* - 1, \quad k^* \leq k.$$

Constraint (A.29) is because job j_{i_1} is released, by its definition, at time $t = 0$. Constraint (A.30) is because job j_{i_ℓ} interferes with a job $j_{i_{\ell+1}}$; the necessary constraint for such an interference is that job $j_{i_{\ell+1}}$ is released before j_{i_ℓ} is completed. Finally, constraint (A.31) assumes that job j_{i_ℓ} does not miss its deadline, for any $i_\ell < k$. (We recall the assumption that tasks $\tau_1, \ldots, \tau_{k-1}$ are schedulable.)

We are now ready to conduct the proofs for (A.26)-(A.28).

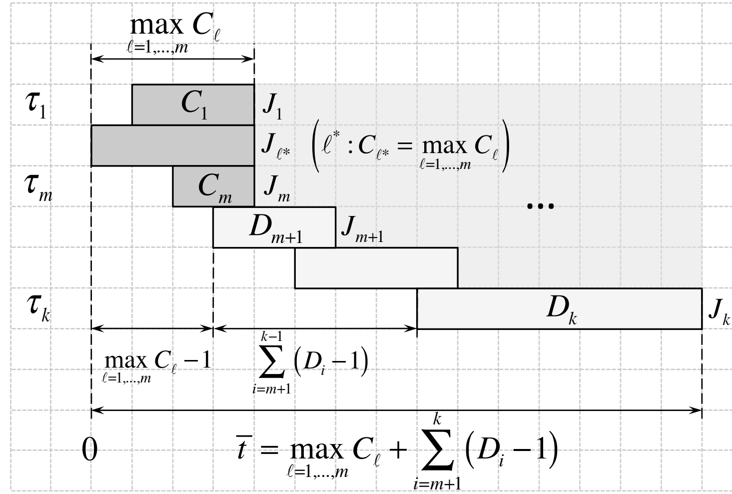


Figure A.2: A tighter upper bound $\bar{t}$ on the length of the longest schedule, satisfying Corollary 1. Such a bound $\bar{t}$ explores the number of available processors m. The figure assumes discrete time.

Proof for Eq. (A.26). Considering (A.29)-(A.31), task τ_k should release a job before time instant $t = \sum_{i=1}^{k-1} D_i$. Indeed,

$$
\begin{aligned}
t_k^{\mathsf{r}} \overset{(A.30)}{<} t_{i_{k^*-1}}^{\mathsf{c}} \overset{(A.31)}{\leq} t_{i_{k^*-1}}^{\mathsf{r}} + D_{i_{k^*-1}} &< t_{i_{k^*-2}}^{\mathsf{c}} + D_{i_{k^*-1}} \\
&\leq \left(t_{i_{k^*-2}}^{\mathsf{r}} + D_{i_{k^*-2}}\right) + D_{i_{k^*-1}} < \ldots < \sum_{\ell=1}^{k^*-1} D_{i_\ell},
\end{aligned}
$$

where t_k^{r} denotes the activation time for task τ_k (see the definition of I above), and $t_{i_1}^{\mathsf{r}} = 0$ due to Eq. (A.29). The right-hand side of the inequality above cannot exceed $\sum_{i=1}^{k-1} D_i$, as $k^* \leq k$ and $i_\ell < i_{\ell+1}$, meaning that (see also Fig. A.1):

$$
t_k^{\mathsf{r}} < \sum_{i=1}^{k-1} D_i.
$$

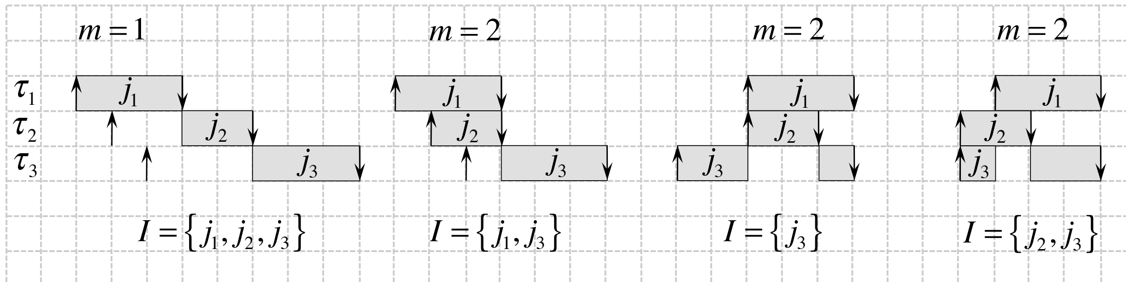


Figure A.3: A sequence of interfering jobs I: cases

As the deadline for a job of task τ_k is $t_k^{\mathrm{r}} + D_k$, the inequality above yields the following bound $\bar{t}$:

$$\bar{t} \quad = \quad \sum_{i=1}^{k} D_i.$$

Eq. (A.26) follows. □

The release scenario, originating (A.26), is depicted in Fig. A.1. Observe that (A.26) does not depend on the number of available processors m.

The following bound is a particular case of (A.26):

$$\bar{t} \quad = \quad t_i^{\mathrm{r}} + \sum_{\ell=i}^{k} D_\ell, \tag{A.32}$$

where t_i^{r} denotes the activation time of an arbitrary task τ_i, $i = 1, \ldots, k$. The proof can be conducted by analogy to Eq. (A.26). We will use (A.32) to prove Eq. (A.27).

We are now ready to prove Eq. (A.27).

Proof for Eq. (A.27). Let us redefine jobs $I = \{j_{i_1}, \ldots, j_{i_{m+1}}\}$ as follows. Job j_{i_ℓ} denotes the first job released by task τ_{i_ℓ}, $\ell = 1, \ldots, m+1$. Jobs in I are sorted in the order of non-decreasing release times. If jobs j_{i_ℓ} and $j_{i_{\ell+1}}$ are released at the same time, then j_{i_ℓ} has a higher priority than $j_{i_{\ell+1}}$ (that is $i_\ell < i_{\ell+1}$). For each job j_{i_ℓ}, let time $t_{i_\ell}^{\mathrm{r}}$ denote its respective release time.

Observe that none of jobs $j_{i_1}, \ldots, j_{i_m}$ experience any interference, until job $j_{i_{m+1}}$ is released. That is each job j_{i_ℓ}, for $\ell \leq m$, will be completed in C_{i_ℓ} time units only, if being executed without a job j_{i_ℓ} being released. Then, the release time $t_{i_{m+1}}^{\mathrm{r}}$ for a job $j_{i_{m+1}}$ should be constrained by

$$t_{i_{m+1}}^{\mathrm{r}} \quad < \quad \max_{\substack{\ell \in \{i_1, \ldots, i_{m+1}\} \\ \ell \neq \max(i_1, \ldots, i_{m+1})}} C_\ell, \tag{A.33}$$

otherwise some job j_{i_ℓ}, $\ell \leq m$, will violate Corollary 1, because job j_{i_ℓ} will be completed without interfering with another job. In the equation above, index $\max(i_1, \ldots, i_{m+1})$ corresponds to a lower priority job $j_{\max(i_1, \ldots, i_{m+1})}$, which is interfered at time $t_{i_{m+1}}^{\mathrm{r}}$.

Considering (A.32) and (A.33), $\bar{t}$ is such that

$$\begin{aligned} \bar{t} \quad &= \quad t_{i_{m+1}}^{\mathrm{r}} + \sum_{\ell=\max(i_1, \ldots, i_{m+1})}^{k} D_\ell \\ &< \quad \max_{\substack{\ell \in \{i_1, \ldots, i_{m+1}\} \\ \ell \neq \max(i_1, \ldots, i_{m+1})}} C_\ell + \sum_{\ell=\max(i_1, \ldots, i_{m+1})}^{k} D_\ell. \end{aligned} \tag{A.34}$$

We finally show that the right-hand side of the inequality above cannot exceed

$$\max(C_1, \ldots, C_m) + \sum_{\ell=m+1}^{k} D_\ell.$$

We prove it by contradiction. Suppose that exists such a case $(i_1, \ldots, i_{m+1})$, different from $(1, \ldots, m+1)$, that an opposite holds:

$$\max(C_1, \ldots, C_m) + \sum_{\ell=m+1}^{k} D_\ell < \max_{\substack{\ell \in \{i_1, \ldots, i_{m+1}\} \\ \ell \neq \max(i_1, \ldots, i_{m+1})}} C_\ell + \sum_{\ell=\max(i_1, \ldots, i_{m+1})}^{k} D_\ell. \tag{A.35}$$

After rearranging the terms, we get that

$$\sum_{\ell=m+1}^{\max(i_1, \ldots, i_{m+1})-1} D_\ell < \max_{\substack{\ell \in \{i_1, \ldots, i_{m+1}\} \\ \ell \neq \max(i_1, \ldots, i_{m+1})}} C_\ell - \max(C_1, \ldots, C_m), \tag{A.36}$$

where $\max(i_1, \ldots, i_{m+1}) > m+1$ due to the assumption $(i_1, \ldots, i_{m+1}) \neq (1, \ldots, m+1)$ and $i_\ell < i_{\ell+1}$.

As the left-hand side of the inequality above is positive, its right-hand side should be also positive. It is achieved if

$$\max_{\substack{\ell \in \{i_1, \ldots, i_{m+1}\} \\ \ell \neq \max(i_1, \ldots, i_{m+1})}} C_\ell > \max(C_1, \ldots, C_m).$$

Let ℓ^* denote the index of a task τ_{ℓ^*}, such that the left-hand side of the inequality above equals to C_{ℓ^*}:

$$\max_{\substack{\ell \in \{i_1, \ldots, i_{m+1}\} \\ \ell \neq \max(i_1, \ldots, i_{m+1})}} C_\ell = C_{\ell^*}. \tag{A.37}$$

These necessary constraints for ℓ^* follow:

$$\begin{aligned} \max_{\substack{\ell \in \{i_1, \ldots, i_{m+1}\} \\ \ell \neq \max(i_1, \ldots, i_{m+1})}} C_\ell = C_{\ell^*} &\Rightarrow \ell^* \leq \max(i_1, \ldots, i_{m+1}) - 1 \\ C_{\ell^*} > \max(C_1, \ldots, C_m) &\Rightarrow \ell^* \geq m+1. \end{aligned} \tag{A.38}$$

Substituting (A.37) into (A.36), we get that

$$\sum_{\ell=m+1}^{\max(i_1, \ldots, i_{m+1})-1} D_\ell < C_{\ell^*} - \max(C_1, \ldots, C_m); \tag{A.39}$$

the necessary condition for this is

$$\sum_{\ell=m+1}^{\max(i_1, \ldots, i_{m+1})-1} D_\ell < C_{\ell^*},$$

which together with constraint (A.38) on ℓ^* yields the following contradiction:

$$\begin{aligned} D_{m+1} + \ldots + D_{\ell^*} + \ldots + D_{\max(i_1, \ldots, i_{m+1})-1} &< C_{\ell^*} \\ D_{\ell^*} &< C_{\ell^*}. \end{aligned}$$

Thus, the assumption (A.35) is infeasible, and (A.27) holds always.

□

We finally prove Eq. (A.28).

Proof for Eq. (A.28). For each task τ_i, with $i < k$, let R_i denote its worst-case execution time, if known, otherwise we set $R_i = D_i$. (We assume that each task τ_i, for $i < k$, is schedulable.)

In the proof for Eq. (A.26) above, we update Eq. (A.31) by replacing term D_{i_ℓ} with R_{i_ℓ}, for $i_\ell < k$. Then, repeating the steps of that proof, we confirm that Eq. (A.26) is reduced to

$$\bar{t} = \sum_{i=1}^{k-1} R_i + D_k,$$

and consequently, Eq. (A.32) is reduced to

$$\bar{t} = t_i^{\mathrm{r}} + \sum_{\ell=i}^{k-1} R_\ell + D_k,$$

where t_i^{r} denotes the activation time of an arbitrary task τ_i. Finally, in the proof for Eq. (A.27), we substitute the equation above into (A.34), instead of (A.32), and we conclude that Eq. (A.28) holds.

□

A.3 Proof of Theorem 2

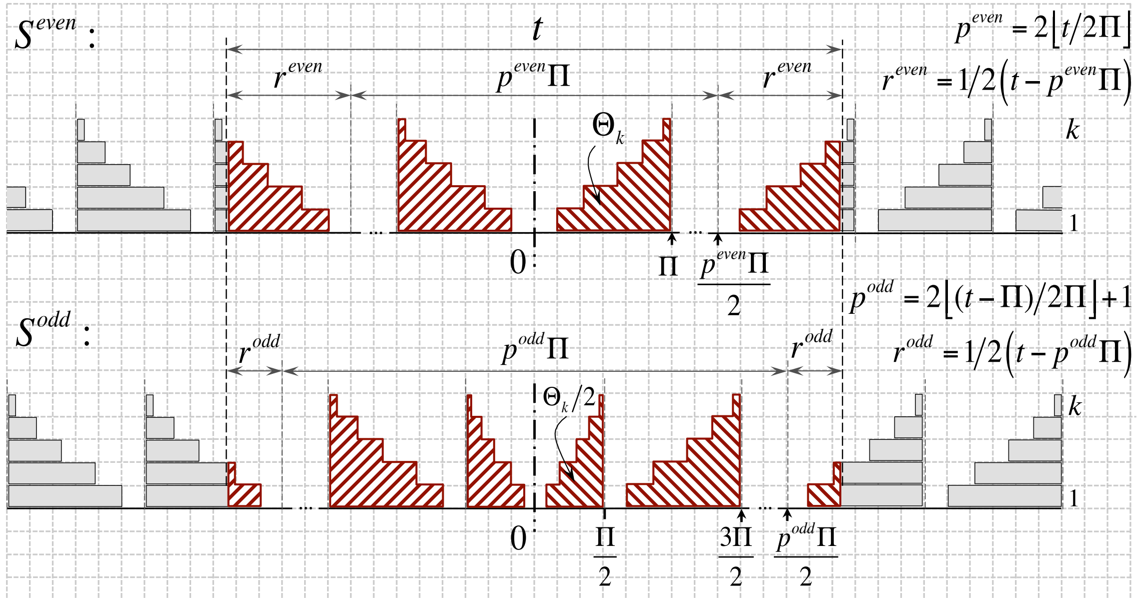


Figure A.4: Theorem 2: GMPR worst-case resource patterns S^{even} and S^{odd}

Proof. Let $\mathsf{supply}_k(S,t)$ denote the resource provided by an arbitrary scenario S in the time interval $\left[-\frac{t}{2},\frac{t}{2}\right]$ (of length t) at concurrency k. We next consider two cases depending on the interval length t: $t \leq \Pi$ and otherwise.

We recall that, from Definition 5 of GMPR, there always exists a time instant t^* such that the resource provided at concurrency k over each interval $[t^*+(p-1)\Pi,\ t^*+p\Pi]$, $p \in \mathbb{Z}$, equals to Θ_k. We refer t^* as the replenishment instant of a GMPR interface, and the time intervals $[t^*+(p-1)\Pi,\ t^*+p\Pi]$ are its replenishment cycles.

Case 1: $t \leq \Pi$. There always exists a replenishment instant $t^* \in \left[-\frac{\Pi}{2},\frac{\Pi}{2}\right]$ such that the resource provided in both intervals $[t^*-\Pi,t^*]$ and $[t^*,t^*+\Pi]$ is Θ_k each. Let us assume that $t^* \geq 0$; the proof for $t^* < 0$ is done by analogy.

As $t^* \in [0,\frac{\Pi}{2}]$ and $\frac{t}{2} \in [0,\frac{\Pi}{2}]$, the following two cases are possible:

$$0 \leq t^* \leq \frac{t}{2} \leq \frac{\Pi}{2}$$
$$0 \leq \frac{t}{2} \leq t^* \leq \frac{\Pi}{2}.$$

Each of these cases is considered below.

Case 1a: $0 \leq t^* \leq \frac{t}{2} \leq \frac{\Pi}{2}$. Let us transform the scenario S into S' by moving left any resource provided before t^* and by moving right any resource provided after t^*, as depicted in Fig. A.5. Since $t^* \in [-\frac{t}{2},\frac{t}{2}]$, such a transformation can only move the resource out of the time interval $[-\frac{t}{2},\frac{t}{2}]$, so that

$$\mathsf{supply}_k(S,t) \geq \mathsf{supply}_k(S',t).$$

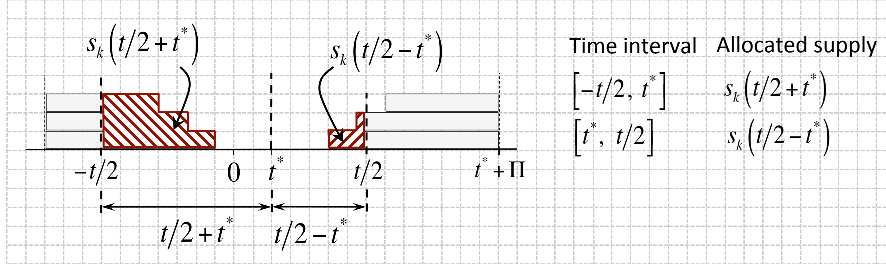


Figure A.5: Case 1(a): $0 \leq t^* \leq \frac{t}{2} \leq \frac{\Pi}{2}$

To analyze the resource supply over S', we now employ the auxiliary function $s_k(t)$ introduced in Lemma 1. From Fig. A.5, it follows that

$$\mathsf{supply}_k(S',t) = s_k\left(\frac{t}{2}+t^*\right) + s_k\left(\frac{t}{2}-t^*\right).$$

Applying condition (3.3) to the RHS of the equation above, we get that

$$\mathsf{supply}_k(S',t) \geq 2s_k\left(\frac{t}{2}\right) = \mathsf{supply}_k\left(S^{\mathsf{even}},t\right),$$

where S^{even} is the resource pattern depicted in Fig. A.4.

Case 1b: $0 \le \frac{t}{2} \le t^* \le \frac{\Pi}{2}$. Let us transform the scenario S into S' by moving out of the time interval $[-\frac{t}{2}, \frac{t}{2}]$ as much resource as possible (see Fig. A.6), so that

$$\mathsf{supply}_k(S,t) \ge \mathsf{supply}_k(S',t).$$

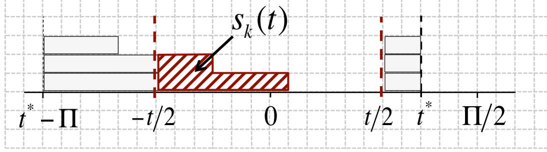


Figure A.6: Case 1(b): $0 \le \frac{t}{2} \le t^* \le \frac{\Pi}{2}$

From Fig. A.6, it follows that

$$\mathsf{supply}_k(S',t) \ge s_k(t) = s_k(t) + s_k(0) \ \ge \ 2s_k\left(\frac{t}{2}\right) = \mathsf{supply}_\mathsf{k}\left(S^{\mathsf{even}},t\right),$$

where the inequality holds due to Lemma 1.

The proof for $t^* \le 0$ is done by analogy to *cases 1a, 1b*. Thus, S^{even} is the worst-case scenario for any $t \le \Pi$.

Case 2: $t > \Pi$. From any scenario S of resource supply, let us transform it into S' by moving left any resource provided before time instant 0 and by moving right any resource provided after 0. Since such a transformation can only move the resource out of the interval, it must again be that

$$\mathsf{supply}_k(S,t) \ge \mathsf{supply}_k(S',t).$$

For S', let us decompose the interval $[-\frac{t}{2}, \frac{t}{2}]$ into the three sub-intervals $[-\frac{t}{2}, t^*]$, $[t^*, t^* + p\Pi]$, and $[t^* + p\Pi, \frac{t}{2}]$ as shown in Fig. A.7, where t^* denotes the first replenishment instant after $-\frac{t}{2}$, and $p \in \mathbb{N}_0$ is the number of full replenishment cycles in $[-\frac{t}{2}, \frac{t}{2}]$.

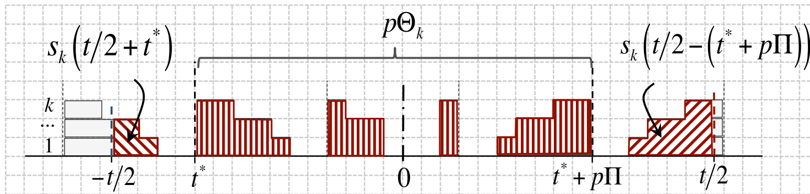


Figure A.7: Case 2: $t > \Pi$

It follows that

$$p \in \left\{ \left\lfloor \frac{t - \Pi}{\Pi} \right\rfloor, \left\lfloor \frac{t}{\Pi} \right\rfloor \right\},$$

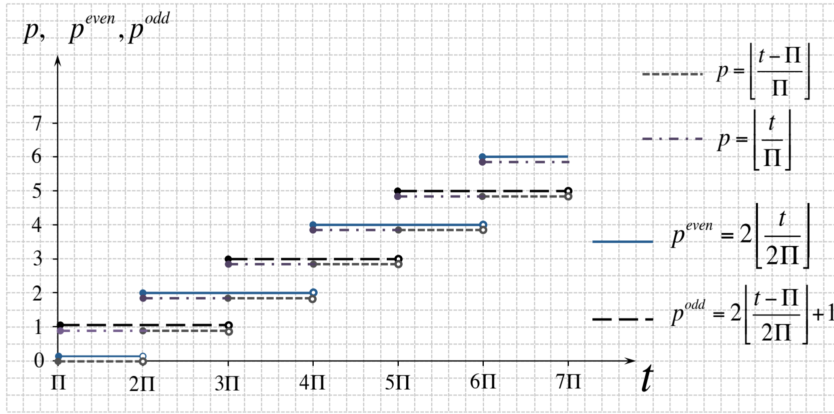


Figure A.8: Comparison of replenishment cycle counters $p, p^{\mathsf{even}}, p^{\mathsf{odd}}$.

which can also be written as $p \in \{p^{\mathsf{even}}, p^{\mathsf{odd}}\}$ (see Fig. A.8 for a graphical interpretation), with

$$p^{\mathsf{even}} = 2\left\lfloor \frac{t}{2\Pi} \right\rfloor \qquad\qquad p^{\mathsf{odd}} = 2\left\lfloor \frac{t-\Pi}{2\Pi} \right\rfloor + 1.$$

The resource $\mathsf{supply}_k(S', t)$ in the interval $[-\frac{t}{2}, \frac{t}{2}]$ is the sum of resource available over the three considered sub-intervals (see Fig. A.7), so that

$$\begin{aligned}\mathsf{supply}_k(S', t) &= s_k\left(\frac{t}{2} + t^*\right) + p\,\Theta_k + s_k\left(\frac{t}{2} - (t^* + p\Pi)\right)\\ &\geq p\,\Theta_k + 2s_k\left(\frac{t - p\Pi}{2}\right),\end{aligned}$$

where the inequality holds due to Lemma 1. In case $p = p^{\mathsf{even}}$, then the equation above turns into

$$\mathsf{supply}_k(S', t) \geq \mathsf{supply}_k(S^{\mathsf{even}}, t),$$

otherwise, if $p = p^{\mathsf{odd}}$, then

$$\mathsf{supply}_k(S', t) \geq \mathsf{supply}_k(S^{odd}, t).$$

Thus, we conclude that no other scenario S exists providing less resource than S^{even} and S^{odd}.

□ □

References

Luís Almeida, Paulo Pedreiras, and José Alberto G. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transaction on Industrial Electronics*, 49(6):1189–1201, December 2002.

Giorgio Ausiello, Pierluigi Crescenzi, Viggo Kann, Giorgio Gambosi, Alberto Marchetti-Spaccamela, and Marco Protazi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 2008.

T. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. *Principles of Distributed Systems, Lecture Notes in Computer Science*, 2007.

Theodore Baker and Sanjoy Baruah. Sustainable multiprocessor scheduling of sporadic task systems. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, 2009.

Sanjoy Baruah. Techniques for multiprocessor global schedulability analysis. In *Proceedings of the IEEE Real-Time Systems Symposium*, 2007.

Sanjoy Baruah and Kirk Pruhs. Open problems in real-time scheduling. *Journal of Scheduling*, 13(6):577–582, 2010.

Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. Improved multiprocessor global schedulability analysis. *Real-Time Systems Journal*, 46:3–24, 2010.

Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, April 2009.

Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, November 2004.

Enrico Bini, Marko Bertogna, and Sanjoy Baruah. Virtual multiprocessor platforms: Specification and use. In *Proceedings of the* 30th *IEEE Real-Time Systems Symposium*, pages 437–446, Washinghton, DC, USA, December 2009.

Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. *Algorithmica*, 2012.

Artem Burmyakov, Enrico Bini, and Eduardo Tovar. The generalized multiprocessor periodic resource interface model for hierarchical multiprocessor scheduling. In *Proceedings of the* 20th *International Conference on Real-Time and Network Systems (RTNS)*, pages 131–139, November 2012.

Yang Chang, Robert Davis, and Andy Wellings. Schedulability analysis for a real-time multiprocessor system based on service contracts and resource partitioning. Technical Report YCS 432, University of York, 2008. available at http://www.cs.york.ac.uk/ftpdir/reports/2008/YCS/432/YCS-2008-432.pdf.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 3 edition, 2009.

Rob Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 2011a.

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, 2011b.

Zhong Deng and Jane win-shih Liu. Scheduling real-time applications in Open environment. In *Proceedings of the* 18th *IEEE Real-Time Systems Symposium*, pages 308–319, San Francisco, CA, U.S.A., December 1997.

Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the* 28th *IEEE International Real-Time Systems Symposium*, pages 129–138, Tucson, AZ, USA, 2007. IEEE Computer Society. ISBN 0-7695-3062-1. doi: http://dx.doi.org/10.1109/RTSS.2007.17.

Xiang Feng and Aloysius K. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the* 23rd *IEEE Real-Time Systems Symposium*, pages 26–35, Austin, TX, U.S.A., December 2002.

Nathan Fisher and Farhana Dewan. Approximate bandwidth allocation for compositional real-time systems. In *Proceedings of the* 21st *Euromicro Conference on Real-Time Systems*, pages 87–96, Dublin, Ireland, July 2009.

Gilles Geeraerts, Joël Goossens, and Markus Lindström. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm. *Real-Time Systems*, 2013.

Nan Guan, Zonghua Gu, Qingxu Deng, Shuaihong Gao, and Ge Yu. Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. *Springer Lecture Notes in Computer Science*, 4761:263–272, 2007.

Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *RTSS*, 2009.

Philip Holman and James H. Anderson. Group-based pfair scheduling. *Real-Time Systems*, 32 (1–2):125–168, February 2006.

Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, October 1986.

Nima Moghaddami Khalilzad, Moris Behnam, and Thomas Nolte. Exact and approximate supply bound function for multiprocessor periodic resource model: Unsynchronized servers. In *Proceedings of CRTS 2012*, December 2012.

Tei-Wei Kuo and Ching-Hui Li. Fixed-priority-driven open environment for real-time applications. In *Proceedings of the* 20th *IEEE Real-Time Systems Symposium*, pages 256–267, Phoenix, AZ, U.S.A., December 1999.

Tei-Wei Kuo, K. Lin, and Y. Wang. An open real-time environment for parallel and distributed systems. In *Proceedings of the* 20th *International Conference on Distributed Computing Systems*, pages 206–213, Taipei, Taiwan, April 2000.

Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the* 20th *Euromicro Conference on Real-Time Systems*, pages 191–200, Prague, Czech Republic, July 2008.

Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Proceedings of the* 15th *Euromicro Conference on Real-Time Systems*, pages 151–158, Porto, Portugal, July 2003.

Giuseppe Lipari and Enrico Bini. A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation. In *Proceedings of the IEEE Real-Time Systems Symposium*, 2010.

C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM (JACM)*, 20, January 1973.

Kimbal Marriott and Peter Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.

Robert McGill, John W. Tukey, and Wayne A. Larsen. Variations of boxplots. *The American Statistician*, 32:12–16, February 1979.

Clifford W. Mercer, Stefan Savage, and Hydeyuki Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, Boston, MA, U.S.A., May 1994.

Mark Moir and Srikanth Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *Proceedings of the* 20th *IEEE Real-Time Systems Symposium*, pages 294–303, Phoenix, AZ, U.S.A., December 1999.

Abday K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, pages 2–13, December 2003.

Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. *Proceedings of the 20th Euromicro Conference on Real-Time Systems conference (ECRTS'08)*, 2008.

Ion Stoica, Hussein Abdel-Wahab, Kevin Jeffay, Sanjoy K. Baruah, Johannes E. Gehrke, and Charles Gregory Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceeding of the* 17th *IEEE Real Time System Symposium*, pages 288–299, Washington, DC, U.S.A., December 1996.

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*, 2014.

Publications

This PhD thesis has resulted in the following publications:

- Artem Burmyakov, Enrico Bini, and Eduardo Tovar. The Generalized Multiprocessor Periodic Resource Interface Model for Hierarchical Multiprocessor Scheduling. *20th International Conference on Real-Time and Network Systems (RTNS 2012), ACM*, 8 to 9, Nov, 2012, pp 131-139. Pont à Mousson, France. **Nominated for the best paper award.**
- Artem Burmyakov, Enrico Bini, Eduardo Tovar. Compositional Multiprocessor Scheduling: the GMPR interface. *Real-Time Systems, Springer US.* May 2014, Volume 50, Issue 3, pp 342-376. U.S.A. **Invited paper.**
- Artem Burmyakov, Enrico Bini, Eduardo Tovar. An exact schedulability test for global FP using state space pruning. *23th International Conference on Real-Time and Network Systems (RTNS 2015), ACM, 4-6th November 2015, pp 225-234. Lille, France*

Acknowledgements

I thank my advisors, Prof. Dr. Eduardo Tovar and Dr. Vincent Nelis, for provided guidance and funding support. I especially thank Prof. Dr. Enrico Bini for a significant help in deriving the results presented in this thesis. I also thank my friends and family for supporting me during these years.

This work was partially supported by FCT (Portuguese Foundation for Science and Technology) and by ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/71368/2010.

Artem Burmyakov