Scheduling of Dependent Tasks with Probabilistic Execution Times on Multi-core Processors

Slim Ben Amor

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ SORBONNE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Slim BEN AMOR

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ SORBONNE

Sujet de la thèse :

Ordonnancement des Tâches avec des Dépendances et des Temps d'Exécution Probabilistes sur Processeur Multi-cœurs

---

soutenue le: 14/12/2020

devant le jury composé de :

Mme. Alix Munier-Kordon	Présidente
Mme. Maryline Chetto	Rapporteuse
M. Emmanuel Grolleau	Rapporteur
M. Sanjoy Baruah	Examinateur
M. George Lima	Examinateur
Mme. Liliana Cucu-Grosjean	Directrice de thèse

This thesis is dedicated
To the soul of my mother who was always ready
to sacrifice everything for us,
To my father and my sisters for their trust and advice, and
To my wife for her endless support and love

Acknowledgements

First, I would like to thank my supervisor, *Liliana CUCU-GROSJEAN*, for her trust and for offering me the opportunity of pursuing PhD studies in the Real-time and embedded systems research domain. Her valuable advice and support when discussing my raw ideas and working on new papers helped me to move forward and accomplish my PhD. At the human level, she has been always a sympathetic and enjoyable person to work with, which made my PhD more pleasant.

I am also immensely grateful to my supervisor and to *Yves SOREL*, a research director at INRIA Paris, and my former teacher, for welcoming me into their research team, as an intern, 4 years ago. They offered to me my first experience in scientific research and they encouraged me to go through a PhD journey that has allowed me to acquire scientific maturity. I also would like to thank *Yasmina ABDEDDAÏM* for offering me the opportunity to teach a machine learning course for engineering students. This experience enabled me to enhance my scientific knowledge as well as my social skills.

In addition, I would like to thank *Emmanuel GROLLEAU* and *Maryline CHETTO* for accepting to review my thesis. Their remarks were really interesting and valuable. I also express my gratitude to the rest of my PhD examining committee: *Alix MUNIER-KORDON*, *Sanjoy BARUAH* and *George LIMA* for their time and interest in my work. I would like to extend special thanks to *George* for the discussions we had and for his advice and encouragement during his visit at INRIA Paris.

Then, I would like to thank my colleagues: *Adriana GOGONEL*, *Cristian MAXIM*, *Walid TALABOULMA*, *Evariste NTARYAMIRA* and *Kevin ZAGALO* with whom I shared an office over these last 4 years. Our every day discussions allowed me to extend my scientific knowledge and to find and refine new ideas. They also helped me to improve my language skills. We spent real pleasant, and sometimes funny moments together. I extend my thanks to all the members of our research team and all the people I met at INRIA Paris during my PhD who made my journey more beneficial and amusing. Huge thanks go to *Roberto MEDINA* and *Richard JAMES* for their help in reviewing some parts of my thesis and papers.

Finally, I would like to thank my parents and sisters for their continuous encouragement and all their sacrifices that allowed me to attain a good educational

level and to accomplish this achievement. My heartfelt thanks go to my wife, who joined me a few month after the start of my PhD journey. The life of a couple who are PhD students can sometimes be complicated with the pressures of work and consecutive deadlines, but she made it pleasant through her endless care, patience, support and love.

Résumé

L'utilisation de plus en plus répandu des processeurs multicœurs dans les systèmes cyber-physiques (CPS) offre des capacités de calcul plus élevées. Il encourage également le développement d'applications plus lourde avec des fonctionnalités interactives et probablement avec des exécutions en parallèles. Souvent, ces applications doivent respecter des contraintes de précédences et d'autres temporelles afin de garantir la validité fonctionnelle et temporelle.

Bien que les architectures multicœurs améliorent en moyenne les performances d'un système, elles introduisent plusieurs incertitudes sur le comportement temporelle. Ces incertitudes sont causées par des communications et des interférences supplémentaires entre les différents cœurs. La plupart des techniques d'analyse en temps réel des systèmes critiques sont basées sur le raisonnement au pire cas. Ils ne prennent en compte que les valeurs au pire cas des paramètres temporelles même si celles-ci se produisent rarement. Par conséquent, l'application de telles techniques pour vérifier le respect des contraintes temporelles conduit à un surdimensionnement des ressources de calcul nécessaires et à un surcoût.

Cependant, les approches d'analyse basées sur le modèle probabiliste prennent en compte toutes les valeurs possibles avec leurs probabilités d'occurrences correspondantes et fournissent la probabilité de dépasser une échéance qui doit rester en dessous d'un seuil donné (par exemple 10^{-7} or 10^{-9}) dépendant de la criticité du système. Le respect de ce seuil permet d'assurer la faisabilité du système avec un niveau de confiance suffisamment élevé tout en évitant le surdimensionnement des ressources.

Dans cette thèse, nous nous intéressons au problème d'ordonnancement des tâches temps réel dures représentées par un graphe orienté acyclique (DAG) chacune et exécutées sur un processeur multicœur. Ces tâches DAG sont indépendantes les unes des autres mais elles présentent des dépendances internes (contraintes de précédence) entre les sous-tâches qui les composent. Nous nous concentrons sur des algorithmes d'ordonnancement partitionnés, à priorité fixe et préemptifs caractérisés par leur comportement statique afin de réduire la variabilité et les interférences entre les différents cœurs. Nous étudions un modèle de tâche qui représente un paramètre temporel avec une seule valeur déterministe égale à sa valeur au pire cas. En outre, nous considérons un modèle de tâche probabiliste qui décrit les différentes valeurs possibles d'un paramètre temporel en utilisant une distribution de probabilité.

Tout d'abord, nous étudions l'ordonnançabilité d'un ensemble de tâches DAG avec des paramètres temporels déterministes. Nous proposons une analyse du temps de réponse (RTA) inspirée d'une analyse existante pour un ensemble de tâches où chacune est composées de sous-tâches liée en chaîne . Ensuite, nous affinons nos équations de temps de réponse en supprimant les quantités qui pourraient être calculés plusieurs fois et les préemptions successives et irréalisables causées par des tâches plus prioritaires. Ainsi, nous réduisons le pessimisme et la surestimation du temps de réponse d'une tâche.

En suite, nous abordons l'analyse d'ordonnançabilité dans le cas de paramètres temporels probabilistes. Nous étendons nos équations de temps de réponse proposées pour les tâches déterministes à l'aide d'opérateurs probabilistes. Ainsi, nous obtenons la distribution du temps de réponse et nous calculons la probabilité de dépassement d'échéance. De plus, nous utilisons un réseau Bayésien pour modéliser les dépendances possibles entre les différentes distributions de probabilité. Nous appliquons également un test d'ordonnançabilité déterministe combiné avec une représentation en C -espace et un classificateur SVM pour estimer la probabilité d'ordonnançabilité.

Enfin, nous proposons des techniques de d'ordonnançement pour définir les priorités aux niveaux des tâches et des sous-tâches afin de réduire le temps de réponse du système. De plus, nous présentons une heuristique de partitionnement qui attribue chaque sous-tâche à un cœur donné d'une façon à promouvoir les exécutions parallèle et à réduire les communications entre les différents cœurs. Puisque la complexité d'une analyse d'ordonnançabilité dépend du nombre de sous-tâches dans un graphe, nous proposons un algorithme qui réduit la taille d'un DAG en fusionnant certains sous-tâches ensemble sans modifier la structure des contraintes de précédence.

Mots clés : Ordonnancement multicœur, Contraintes de précédence, Probabilité de dépassement d'échéance, Analyse du temps de réponse, Classification SVM, Réseaux bayésiens.

Abstract

The use prevalence of multi-core processors in Cyber-Physical Systems (CPSs) provides greater computation capacities. It also encourages the development of more resource-demanding applications with interactive functionalities and, in most cases, parallel executions. Often, these applications must respect precedence and timing constraints in order to ensure functional and temporal correctness.

Although multi-core architectures enhance the system's performance on average, they introduce some uncertainties about the timing behavior. These uncertainties are caused by additional communications and interferences between different cores. Most real-time analysis techniques for safety-critical systems are based on worst-case reasoning: they consider only worst-case values of timing parameters, even if they occur rarely. Therefore, applying such techniques to verify that timing constraints are respected leads to an over-sizing of computational resources required and to an extra-cost.

However, analysis approaches based on a probabilistic model takes into account all possible values with their corresponding probabilities of occurrences and they provide the probability of missing a deadline that should not exceed a given threshold (e.g. 10^{-7} or 10^{-9}) depending on the criticality of the system. Respecting this threshold makes it possible to ensure the feasibility of the system with a sufficiently high confidence level while avoiding the over-sizing of the resources required.

In this thesis, we are interested in the scheduling problem of hard real-time tasks, each of which is represented by a Directed Acyclic Graph (DAG) and executed on a multi-core processor. These DAG tasks are independent of each other but they present internal dependencies (precedence constraints) between the sub-tasks composing them. We focus on partitioned, fixed-priority and preemptive scheduling algorithms characterized by their static behavior in order to reduce variability and interference between different cores. We study a task model that represents a timing parameter with a single deterministic value equal to its worst-case value. Furthermore, we consider a probabilistic task model that describes different possible values of a timing parameter using a probability distribution.

First, we study the schedulability of DAG task model with deterministic timing parameters. We propose a Response Time Analysis (RTA) inspired by an existing analysis for a task model composed of chains of sub-tasks. Then, we refine our

response time equations by removing delays that could be computed several times and unrealistic successive preemptions of higher priority tasks. Thus, we reduce the pessimism and the over-estimation of the response time of a DAG task.

Second, we address schedulability analysis in the case of probabilistic timing parameters. We extend our response time equations proposed for deterministic tasks using probabilistic operators. Thus, we obtain the response time distribution and we compute the deadline miss probability. Moreover, we use a Bayesian network to model possible existing dependencies between different probability distributions. We also apply a deterministic schedulability test combined with a C-space representation and an SVM classifier to estimate the schedulability probability.

Third, we provide some scheduling techniques to define priority at the task and sub-task levels in order to enhance the reactivity of the system. We also present a partitioning heuristic that assigns each sub-task to a given core in such a way to promote parallel executions and reduce communications between different cores. Since the complexity of a schedulability analysis depends on the number of nodes, we propose an algorithm that reduces the size of a DAG by merging some nodes together without altering the structure of precedence constraints.

Keywords: Multi-core scheduling, Precedence constraints, Deadline miss probability, Response time analysis, SVM classification, Bayesian networks.

List of my Publications

1. Slim BEN-AMOR, Dorin MAXIM, and Liliana CUCU-GROSJEAN. "Schedulability analysis of dependent probabilistic real-time tasks". In *RTNS* 2016.

2. Slim BEN-AMOR, Dorin MAXIM, and Liliana CUCU-GROSJEAN. "Towards schedulability analysis of real-time systems withprecedence constraints and different periods". In *JRWRTC* 2016.

3. Slim BEN-AMOR and Liliana CUCU-GROSJEAN. "Probabilistic parallel real-time tasks model on multiprocessor platform". In *RTSOPS* 2018.

4. Slim BEN-AMOR, Liliana CUCU-GROSJEAN, and Dorin MAXIM. "Response time analysis for precedence constrained and partitioned multiprocessor scheduled tasks". In *JRWRTC* 2018.

5. Slim BEN-AMOR, Liliana CUCU-GROSJEAN, and Dorin MAXIM. "Worst-case response time analysis for partitioned fixed-priority dag tasks on identical processors". In *WIP session ETFA* 2019.

6. Slim BEN-AMOR, Liliana CUCU-GROSJEAN, Mehdi MEZOUAK, and Yves SOREL. "Probabilistic schedulability analysis for real-time tasks with precedence constraints on partitioned multi-core". In *WIP session ISORC* 2020.

7. Slim BEN-AMOR, Liliana CUCU-GROSJEAN, Mehdi MEZOUAK, and Yves SOREL. "Probabilistic schedulability analysis for precedence constrained tasks on partitioned multi-core". In *ETFA* 2020.

Contents

List of Figures

List of Tables

List of Abbreviations

3-SAT . . . . . 3 Satisfiability.

BCET . . . . . Worst-Case Execution Time.

BCRT . . . . . Worst-Case Response Time.

CDF . . . . . Cumulative Distribution Function.

CPT . . . . . Conditional Probability Table.

DAG . . . . . Directed Acyclic Graph.

DMP . . . . . Deadline Miss Probability.

DM . . . . . Deadline Monotonic.

ECM . . . . . Engine Control Module.

ECU . . . . . Electrical Control Unit.

EDF . . . . . Earliest Deadline first.

EVT . . . . . Extreme Value Theory.

GA . . . . . genetic algorithm.

GEV . . . . . Generalized Extreme Value.

GPD . . . . . Generalized Pareto Distribution.

HyPTA . . . . . Hybrid Probabilistic Timing Analysis.

ILP . . . . . Integer Linear Programming.

MBPTA . . . . Measurement-Based Probabilistic Timing Analysis.

MILP . . . . . Mixed Integer Linear Programming.

MIT . . . . . Minimum Inter-Arrival Time.

OPA . . . . . Optimal Priority Assignment.

PMF . . . . . Probability Mass Function.

PoT . . . . . Peaks-over-Threshold.

pWCCT . . . . probabilistic Worst-Case Communication Time.

pWCET . . . . . probabilistic Worst-Case Execution Time.

pWCRT . . . . . probabilistic Worst-Case Response time.

RM . . . . . Rate Monotonic.

RTA . . . . . Response Time Analysis.

SPTA . . . . . Static Probabilistic Timing Analysis.

SimSo . . . . . Simulation of Multiprocessor Scheduling with Overheads.

SVM . . . . . Support Vector Machine.

VE . . . . . Variable Elimination.

WCCT . . . . . Worst-Case Communication Time.

WCET . . . . . Worst-Case Execution Time.

WCRT . . . . . Worst-Case Response time.

List of Symbols

Symbol	Description
τ	Set of n DAG tasks.
τ_i	DAG task i at priority level i.
$\tau_{i,j}$	Sub-task j belonging to DAG task τ_i.
n_i	Number of sub-tasks composing τ_i.
π	Processor having m identical cores.
π_i	Core i of the processor π.
$\pi(.)$	Function that maps each sub-task to a core.
G_i	DAG associated to τ_i.
D_i	Deadline before which τ_i must complete its execution.
T_i	Minimum inter-arrival time between two consecutive arrivals of τ_i.
V_i	Set of sub-tasks composing τ_i.
E_i	Set of the precedence constraints between the sub-tasks of τ_i.
$e_i(j,k)$	Precedence constraint (edge) between the sub-tasks of $\tau_{i,j}$ and $\tau_{i,k}$.
$C_{i,j}$	WCET of $\tau_{i,j}$.
$hep(\tau_{i,j})$	Set of sub-tasks belonging to other DAGs with higher or equal priority to τ_i.
$hep_i(\tau_{i,j})$	Set of sub-tasks belonging to DAG task τ_i and having higher or equal priority to $\tau_{i,j}$.
$succ(\tau_{i,j})$	Set of all successors of $\tau_{i,j}$.
$isucc(\tau_{i,j})$	set of the immediate successors of $\tau_{i,j}$.
$succ^*(\tau_{i,j})$	Set of all successors of $\tau_{i,j}$ including itself.
$pred(\tau_{i,j})$	Set of all predecessors of $\tau_{i,j}$
$ipred(\tau_{i,j})$	Set of the immediate predecessors of $\tau_{i,j}$.
$pred^*(\tau_{i,j})$	Set of all predecessors of $\tau_{i,j}$ including itself.
$parallel(\tau_{i,j})$	Set of sub-tasks independent of $\tau_{i,j}$
$G^{cnx}_{pred}(\tau_{i,j})$	Set of connected predecessors of $\tau_{i,j}$, executed on the same core as $\tau_{i,j}$.

$w_{i,j}$	Local response time as defined in [1].
$J_{i,j}$	Release jitter of sub-task $\tau_{i,j}$ in the sub-task chain τ_i.
$I_i^{int}(\tau_{i,j})$	Internal interference of $\tau_{i,j}$ caused by parallel sub-tasks from τ_i.
$I^{ext}(\tau_{i,j})$	External interference of $\tau_{i,j}$ caused by sub-tasks belonging to any higher priority DAG task.
$I^{cnx}(\tau_{i,j})$	External interference exerted on $G_{pred}^{cnx}(\tau_{i,j})$.
$I_{i,j}^{pred}(\tau_{i,k})$	Interference on $\tau_{i,k}$ and its predecessors caused by other predecessors of $\tau_{i,j}$.
$R_{i,j}^{isol}$	Response time in isolation of $\tau_{i,j}$ (Response time considering sub-tasks from the same DAG).
$R_{i,j}^{seq}$	Sequential response time of $\tau_{i,j}$ (Response time considering only longest predecessors sequence).
$R_{i,j}^{pred}$	Preceding response time of $\tau_{i,j}$ (Response time considering all predecessors).
$R_{i,j}^{global}$	Global response time of $\tau_{i,j}$ (Response time considering all DAG tasks).
R_i^{global}	Global response time of the whole DAG τ_i.
$P_{i,j}$	Set of sub-tasks from the same DAG that could preempt $\tau_{i,j}$.
$\Pi_{i,j}$	Set of parallel sub-tasks that could preempt one of the sub-tasks in $pred^{\star}(\tau_{i,j})$.
$\Pi_{i,j}^{pred}$	Set of sub-tasks not predecessors to $\tau_{i,j}$ that could preempt $\tau_{i,j}$ or one of its predecessors.
$\Pi_{i,j}^{cnx}$	Set of sub-tasks parallel to a sub-task in $G_{pred}^{cnx}(\tau_{i,j})$ executing on the same core and having higher or equal priority.
$\Psi_{i,j}(\tau_{i,k})$	Set of predecessors to $\tau_{i,j}$ and not to $\tau_{i,k}$ that could preempt a sub-task in $pred^{\star}(\tau_{i,k})$.
$\mathcal{C}_{i,j}$	Discrete random variable characterizing the pWCET of $\tau_{i,j}$.
$K_{\mathcal{C}_{i,j}}$	Set of possible values of $\mathcal{C}_{i,j}$.
$f_{\mathcal{C}_{i,j}}$	Probability mass function of $\mathcal{C}_{i,j}$.
$F_{\mathcal{C}_{i,j}}$	Cumulative distribution function of $\mathcal{C}_{i,j}$.
$\mathbb{E}(\mathcal{C}_{i,j})$	Expected value of $\mathcal{C}_{i,j}$ distribution.
$\mathcal{E}_i(j,k)$	pWCCT between sub-tasks $\tau_{i,j}$ and $\tau_{i,k}$.
DMP_i	Deadline Miss Probability of τ_i.
$\otimes$	Convolution operator.
$\ominus$	Complementary operator of convolution.
$\mathcal{M}\text{ax}_{\text{Indep}}$	Probabilistic maximum operator based on independent random variables.
$\mathcal{M}\text{ax}_{\text{Diaz}}$	Probabilistic maximum operator based on the Diaz[2] comparison.
$\mathcal{M}\text{ax}_{\text{Copula}}$	Probabilistic maximum operator based on copula bound [3].
$\mathcal{R}_{i,j}^{global}(Diaz)$	Global response time of $\tau_{i,j}$ distribution based on $\mathcal{M}\text{ax}_{\text{Diaz}}$ operator.

1 Introduction

Contents

1.1 Context and Motivation

The increasingly widespread deployment of intelligent devices has led designers of embedded and real-time systems to integrate additional functionalities, thereby rising the complexity of the design and the validation process. Often, a functionality is accomplished by a set of programs (or tasks), related by precedence constraints. These constraints must be satisfied in order to ensure the functional correctness. For example, the internal combustion engines of vehicles manufactured during the last decade are no longer controlled by mechanical systems. In fact, a specific software executed on the Engine Control Module (ECM) manages and triggers each cycle of the engine using different sensors and actuators [4] such as a crankshaft position sensor, fuel injectors and valves. This controller enforces the precedence constraints like valve closing or ignition, which are defined between different parts of the software drivers to achieve the desired functioning and optimal performance. These precedence constraints are often modeled by Directed Acyclic Graphs (DAGs).

On the other hand, chip manufacturers are constantly seeking to improve the performance of their chips while reducing power consumption. Since the clock

frequency is physically limited by the heat release, these companies have incorporated several cores on the same processor to allow simultaneous processing, which offers a speedup for executing programs. For instance, Intel® has proposed the Xeon Phi™ 7920 processor with more than 70 cores [5]. Meanwhile, programming paradigms are also evolving, in order to follow the development of hardware architectures. New parallel programming models have been introduced such as OpenMP [6] and Intel Threading Building Blocks [7]. These models exploit the possible intra-task parallelism by dividing large tasks into smaller sub-tasks and running them in parallel. Then, they synchronize and merge their results. Such an approach creates precedence constraints between several sub-tasks (threads) inside the same task (program). Thus, a DAG task model is adopted to describe different independent programs (or tasks) as well as the dependent threads composing them.

Although in many advanced applications, real-time systems require intensive computation resources, they do not take full advantage of the parallel processing provided by multi-core processors. To the best of our knowledge, current timing analysis techniques do not allow parallel processed systems to be validated and certified. In fact, these techniques are designed to analyze simple software running on simple and predictable architectures. However, multi-core processors constitute complex and unpredictable architectures. They cause additional interference and communication delays between the different tasks executed on different cores, which introduces an important variability on the execution times of tasks [8]. This variability results from the gap between the average and the worst-case execution time.

In general, schedulability analysis deployed for hard real-time systems are based on the study of worst-case scenarios. Hence, they tend to reject the schedulability of tasks with a high variability in the execution time. Actually, such a system is likely to violate timing constraints in the worst case even if it meets them in most cases, which increases the pessimism of the analysis and induces an extra cost for manufacturing and validating the system. For example, if a program invokes an error recovery routine when errors occur, then it will have two values of execution times: a high value when the recovery routine is triggered and a lower value otherwise. A worst-case based reasoning considers only the high value of the execution time. This leads to a pessimistic estimation of the actual response time, especially if errors appear rarely (e.g. failure rate lower than 10^{-7} per hour of operation) and the difference between the two execution time values is large. Based on this analysis, additional computational resources are required in order

to reduce the estimated response time and to pass the schedulability test. But in most cases, this causes a poor utilization of the system resources.

Moreover, the over-sizing of the hardware resources affects not only their cost but also other non-functional constraints like their power consumption, size, weight and heat. These constraints are crucial for some embedded and real-time systems like Unmanned Aerial Vehicles (UAVs) and Autonomous Underwater Vehicles (AUVs). In order to reduce hardware over-sizing while ensuring timing requirements and other non-functional constraints, our work is based on a recent schedulability approach that takes into account the variability of timing parameters like the execution and communication times. We represent these parameters by discrete probability distributions with different possible values. The probabilistic analysis consists in estimating a Deadline Miss Probability (DMP) for each task. If the execution times have large values that are not frequent, then the DMP will not be significantly increased. Hence, the system becomes schedulable with high confidence, i.e. low DMP, instead of being evaluated as non feasible by the worst-case reasoning because of the non frequent large values. We deduce that the probabilistic approach allows us to reduce the pessimism of the schedulability analysis.

Probabilistic analysis is applicable on soft real-time systems to guarantee a high quality of service when the DMP is small. It is also useful for industrial systems following safety standards that require a low probability of failure, such as IEC-61508 [9] and ISO-26262 [10]. Furthermore, we can extend this approach to be applied to safety-critical fields such as avionics. Indeed, we note that the mechanical components of an aircraft are designed with a given small failure rate that must be respected (e.g. lower than 10^{-9} failure per hour of operation). Similarly, software with a very low failure rate (i.e. DMP) could be reasonable for such critical systems [11].

As mentioned previously, interference in multi-core processors increases the variability of tasks' execution times. Moreover, using global and dynamic priority scheduling amplifies this phenomenon because of the continuous modifications of the priority level and the executing core for each sub-task. Hence, the schedulability analysis of hard real-time systems becomes overly pessimistic. However, applying partitioned and fixed-priority scheduling reduces the interference and variability due to their static execution behavior. In this sense, we believe that considering partitioned and fixed-priority represents a more promising approach for hard real-time systems scheduling of DAG tasks on multi-core processors.

1.2 Problem Description and Objectives

In this thesis, we focus on the problem of hard real-time scheduling of DAG tasks on identical multi-core processors using a partitioned, fixed-priority and preemptive scheduling policy. We use DAG tasks to model programs with precedence constraints (data dependencies) and parallel applications with intensive computations. Moreover, we acknowledge the existence of possible communication delays between different related sub-tasks. These delays are caused by data transfers or any other kind of communications. We also recognize the variability of timing parameters like execution times and communication delays. Therefore, we consider a task model characterized by deterministic or even by probabilistic timing parameters.

There are two main research focuses in this thesis:

- the proposition of a safe and sufficient schedulability test that ensures meeting the timing requirements with a high confidence level of schedulability for systems with probabilistic parameters.
- the proposition of scheduling techniques that allow us to reduce the response time and enhance the reactivity of the system.

By using these scheduling and schedulability techniques, we reduce the pessimism and the resources over-sizing while satisfying timing requirements as well as non-functional constraints like cost, power consumption, size and weight.

The work presented in this dissertation is partially funded by the FR FUI22 CEOS project[1] that has ADCIS, RTaW, THALES, DGAC, EDF, ENEDIS and Aéroport de Caen Carpiquet as partners. This project develops a reliable and secure system for automatic inspections of equipment and infrastructure elements using professional mini-drones.

1.3 Thesis Outline

The organization and contents of the chapters of this thesis are presented below.

- In Chapter 2, we present an overview of the state-of-the-art and recent research on the scheduling and schedulability of real-time systems with deterministic and probabilistic timing parameters. First, give definitions of main concepts and terms related to real-time systems. Second, we describe different categories of multi-core and multiprocessor scheduling. Third, we present scheduling

[1] `https://www.ceos-systems.com`

techniques for a parallel task model with precedence constraints. Finally, we specify how probabilistic timing parameters, like execution time, are deployed in real-time scheduling and how they are estimated. We also present several schedulability results for a probabilistic task model for single core and multi-core processors.

- In Chapter 3, we study the schedulability of a DAG task model with deterministic timing parameters using response time analysis. We consider a partitioned, fixed-priority and preemptive scheduling policy on multi-core processors. First, we extend existing response time equations for distributed systems to deal with a DAG task model through three steps. For each step, we provide theoretical proofs to guarantee the safety of our response time estimation. Second, we provide two different methods to characterize worst-case arrival patterns for higher priority tasks in order to reduce the pessimism in the response time estimation.

- In Chapter 4, we tackle the DAG task model with probabilistic timing parameters. We start by proposing a probabilistic maximum operator and we extend the response time equations proposed in Chapter 3 to deal with discrete probability distributions that represent timing parameters and we estimate response time distributions. Then, we use a Bayesian network to model the dependency between different random variables involved in response time equations, which enables us to compute the actual response time distributions. In the last section, we represent probabilistic timing parameters and schedulability condition as regions in the C-space and we combine the deterministic schedulability test (proposed in Chapter 3) and a classification technique from machine learning domain (Support Vector Machine) in order to estimate the schedulability probability.

- In Chapter 5, we provide scheduling techniques that reduce the response times of DAG tasks scheduled with a partitioned, fixed-priority and preemptive policy. In the first section, we provide several methods to define priorities for a DAG task model at the task and at the sub-task levels. In the second section, we propose a partitioning heuristic that assigns each sub-task to a given core while balancing the load between cores. In the third section we describe two graph reduction approaches that reduce the size of a DAG task by merging some nodes together while preserving the original structure of precedence constraints. The last section is dedicated to the presentation of an integrated methodology of applying different scheduling techniques together in order to reduce the response time and enhance the reactivity of the system.

- In Chapter 6, we present the evaluation results of the different scheduling and schedulability techniques proposed in this thesis. First, we apply these techniques on randomly generated task sets with different parameters and we compare these results with results obtained from simulating the task set on a study interval equal to the hyperperiod. Second, we assess probabilistic schedulability techniques on a real use case of a PX4 autopilot used for controlling different types of UAVs and mobile robots.
- In Chapter 7, we summarize our contributions and we present future research perspectives.

2

State Of The Art

Contents

In this chapter, we present related work with respect to the contribution of this thesis. We start by defining terminology and main concepts used in real-time domain. Then, we present the main existing results and categories of multi-core scheduling problems. After that, we focus on scheduling techniques and schedulability of dependent tasks with precedence constraints on both single core and multi-core processors. We dedicate, also, a section to the presentation of existing relevant work with respect to the probabilistic real-time analyses. We illustrate some techniques of probabilistic timing analysis. We also differentiate between probabilistic schedulability analyses on single core and multi-core processors.

2.1 Real-time Domain and Terms Definitions

A real-time system is a reactive system that must respond to an external input or event within a specified time constraint. The correctness of an output delivered by such a system depends not only on the logical or functional correctness but also on the time instant at which the output is produced, i.e. a correct calculation but out of time is an invalid calculation. Real-time systems are present in many applications nowadays such as transportation, industrial automation, medical systems, multimedia, and communications. According to the criticality of timing constraints and the usefulness of the results after the deadline (cf. Figure 2.1), real-time systems can be classified as follows:

- **Hard real-time systems:** The usefulness of the results after the deadline becomes negative or minus infinity according to the criticality of the system. These system experience a failure after a deadline miss. For safety-critical system, missing deadline can lead to a disastrous consequences like life loss and huge economic damage. For instance, avionics, aerospace, nuclear plants are safety-critical systems.

- **Soft real-time systems:** The usefulness of the results decreases after the deadline but remains positive for a given interval of time. This decrease of usefulness represents the tolerance for deadline misses. These systems do not fail immediately after a deadline miss as hard real-time systems and they are often related to a quality-of-service such as multimedia and telecommunication

- **Firm real-time systems:** The usefulness of the results after the deadline becomes equal to 0. These systems are similar to soft real-time systems. They do not fail after a deadline miss but they do not benefit from late delivery of service. We cite financial forecast systems and robotic assembly lines as examples.

2.1.1 Real-time Task Model

The most used task models in real-time system domain is the **periodic task model** proposed by Liu and Layland [12] in 1973. This model defines a real-time application as a set of n independent sequential tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i releases an infinite sequence of identical instances, which are called "jobs". τ_i is characterized by the following parameters:

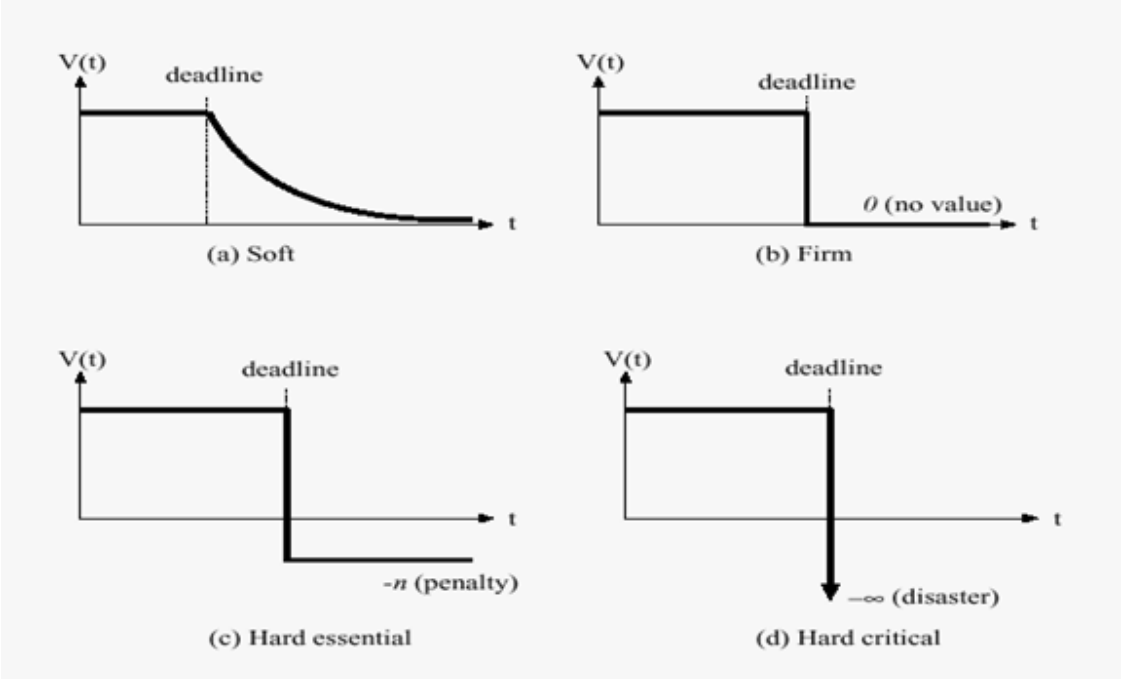


Figure 2.1: Usefulness of results in function of time for different real-time systems.

- **Worst-Case Execution Time (WCET): C_i** an upper bound of the time required by a processor to execute a job of the task τ_i without interruption.
- **Period: T_i** the exact delay between consecutive jobs releases of the task τ_i.
- **Relative deadline: D_i** the delay from the release of a job of the task τ_i before which the job should finish its execution.

These parameters are illustrated in Figure 2.2

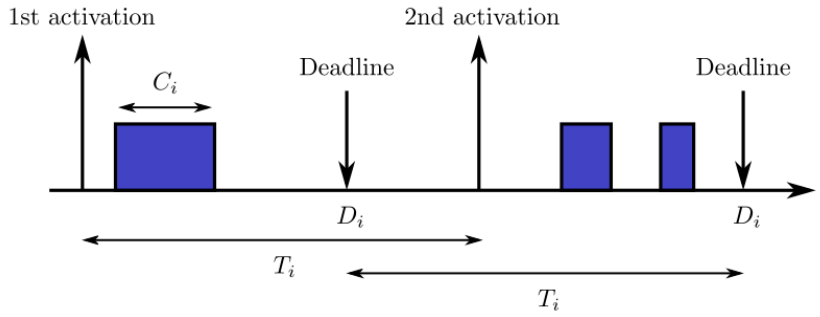


Figure 2.2: The parameters of the real-time task τ_i according to the periodic task model [12].

According to the relation between the period and the deadline, a real-time task τ_i is said to be:

- **Implicit deadline:** if the deadline of τ_i is equal to its period (i.e. $D_i = T_i$).
- **Constrained deadline:** if the deadline of τ_i is less than or equal to its period (i.e. $D_i \leq T_i$).
- **Arbitrary deadline:** if the deadline of τ_i can be less, equal or greater than its period (i.e. there are no constraints between D_i and T_i).

Besides the periodic task model, there is the **sporadic task model** that is similar the periodic one. The only difference is that the delay between consecutive job releases of a task τ_i is greater than or equal to T_i and not exactly equal to T_i as for periodic task model. In a such case, T_i is known as the minimum inter-arrival time.

A task set is referred to as **synchronous** or **asynchronous** based on the first activation scenario of its tasks. A synchronous task set is defined as the task set whose first job of its tasks are activated at the same time. While the first jobs of an asynchronous task set are activated at different times.

The **utilization factor** U_i of a task τ_i is the fraction of the processor's capacity occupied by the jobs of τ_i. It is given as follows:

$$U_i = \frac{C_i}{T_i}$$

Similarly, the utilization of task set τ is defined as the sum of utilization of its tasks, i.e. $U_{sum} = \sum_{i=1}^{n} U_i$.

For a task τ_i, we also define the **density** δ_i with respect to its deadline as follows:

$$\delta_i = \frac{C_i}{\min(D_i, T_i)}$$

.

For a constrained deadline task τ_i, we note that the density is equal to $\delta_i = \frac{C_i}{D_i}$. In the case of an implicit deadline task, the density and utilization are equal $\delta_i = U_i$ since period and deadline are equal ($T_i = D_i$).

2.1.2 Real-time Scheduling

The role of a **real-time scheduler** consists in selecting, from the set of active jobs, the job to execute on each available core at each time instant. The strategy used for selecting jobs and assigning cores is determined by a **real-time scheduling algorithm**. This algorithm should take scheduling decisions that allow every task in the system to respect their timing constraints. Real-time schedulers are commonly divided in two categories:

- **Off-line schedulers:** take scheduling decisions based on a scheduling table pre-computed at the design time. In order to construct a scheduling table, release times and deadlines of all tasks must be known a priori. An example of systems using static scheduling tables are Time-Triggered systems. These approaches make the scheduling completely deterministic and easier to certify. However, off-line schedulers require to generate a new scheduling table after each modification of the system. In addition, they do not deal with sporadic arrivals.

- **On-line schedulers:** take scheduling decisions at the run-time according to the scheduling algorithm used. Most of these algorithms are called *priority-driven* because a priority level is defined for each job. These priorities are updated at each invocation of the scheduler and the scheduling decisions are taken to promote the execution of higher priority jobs and tasks. On-line schedulers are used usually in dynamic et Event-Triggered systems such as sporadic arrival where jobs may arrive at an unknown time.

Remark. *In this thesis, we are interested in systems with an on-line scheduler that executes the scheduling algorithm and makes the scheduling decisions at run-time, which offers more flexibility and tolerance to dynamic behaviors and external events.*

Priority-driven Scheduling

Furthermore, priority-driven scheduling algorithms can also be categorized according to how the job priorities may vary over time as follows:

- **Task-Level Fixed Priority:** each task is assigned a fixed priority based on its timing parameters. All jobs of the same task inherit the same fixed-priority. Thus, the scheduling decisions are not affected by the elapsed time. Examples of such algorithms is the Rate Monotonic [12] (RM) and the Deadline Monotonic [13] (DM) scheduling algorithms.

- **Job-Level Fixed Priority:** each job is assigned a fixed priority, according to its timing parameters at its activation. Hence, different jobs of the same task may have different priorities. An example of this category is the Earliest Deadline First (EDF) algorithm.

- **Job-Level Dynamic Priority:** each job is assigned a priority that may change during its execution. An example of this category is the Least Laxity First (LLF) scheduling algorithm.

Preemptive/non-Preemptive Scheduling

Real-time scheduling algorithms are also classified based on the execution behavior of high priority tasks with respect to lower priority tasks as follows:

- **Preemptive scheduling:** the execution of a running job can be interrupted (preempted) by higher priority jobs. Its execution is resumed only after all the active higher priority jobs are terminated or suspended.
- **Non-preemptive scheduling:** the execution of a running job cannot be interrupted until its completion. The execution of a higher priority job may be delayed by at most one lower priority job. This effect is known as blocking.
- **Cooperative scheduling:** the execution of a running job may only be preempted at defined scheduling points within its execution. Effectively, the execution of each job is composed of a series of preemptable and non-preemptable sections.

Migration for Multi-core Scheduling

Multi-core scheduling algorithms are categorized according to when the change of the core allocation for jobs can be made:

- **No migration:** Each task is allocated statically to a core and no migration is permitted.
- **Task-level migration:** The jobs of the same task may execute on different cores. However, each job can only execute or resume execution (after a preemption) on the same core, on which it starts its execution.
- **Job-level migration:** A single job can execute or resume execution on different cores. However, parallel execution of the same job on different cores at the same time is not permitted.

Remark. *No migration scheduling algorithm category is also known as* ***Partitioned scheduling****. While,* ***Global scheduling*** *refers to permitted migration categories whether on the task or job levels. Results regarding to these two categories are presented in the next Section 2.2.*

A **work-conserving** scheduling algorithm is a scheduling algorithm that schedules active jobs on available cores and does not delay them if there are any idle cores. Partitioned scheduling algorithms are not work-conserving, because an active job could be waiting for its assigned core to be free while other cores are idle but cannot execute it.

A scheduling algorithm is said to be **clairvoyant** if it makes use of information about future events that are not generally known until they happen, such as the precise arrival times of sporadic tasks or actual execution times.

2.1.3 Real-time Schedulability Analysis

Before defining schedulability analysis, we should define some related terms and notions:

A **valid schedule** for a task set τ is a schedule in which all jobs of each task $\tau_i \in \tau$ respect their deadlines.

A **feasible** task set τ is a task set for which a valid schedule could be found.

A task set τ is said to be **schedulable** by a scheduling algorithm $\mathcal{A}$, if a valid schedule could be found using algorithm $\mathcal{A}$. In this case, we say that τ is **$\mathcal{A}$-schedulable**.

An scheduling algorithm $\mathcal{A}$ is said to be **optimal** if it succeed to find a valid schedule for any feasible task set (i.e. any feasible task set is $\mathcal{A}$-schedulable).

Comparing Scheduling Algorithms

The relations between two real-time scheduling algorithms $\mathcal{A}$ and $\mathcal{B}$ could be determined as follows:

- **Dominant:** $\mathcal{A}$ is dominant compared to $\mathcal{B}$, if all task sets that are $\mathcal{B}$-schedulable, are also $\mathcal{A}$-schedulable and there is at least a task set that is schedulable by $\mathcal{A}$ but not by $\mathcal{B}$.

- **Equivalent:** $\mathcal{A}$ and $\mathcal{B}$ are equivalent, if all task sets that are schedulable by one of the two algorithms are also schedulable by the other. This also means that all task sets that are non-schedulable by one of the two algorithms are also non-schedulable by the other.

- **Incomparable:** $\mathcal{A}$ and $\mathcal{B}$ are incomparable, if there are some task sets that are schedulable by $\mathcal{A}$ but not by $\mathcal{B}$. Meanwhile, it exists some task sets that are schedulable by $\mathcal{B}$ but not by $\mathcal{A}$.

In order to compare the efficiency of scheduling algorithms and schedulability analyses, performance metrics are used in literature [14]. Below, we recall some of these metrics:

Definition 2.1. *The **utilization bound** of a scheduling algorithm $\mathcal{A}$, denoted by $U_{\mathcal{A}}$, is defined as the greatest positive number such that: all task sets with implicit deadlines that have a total utilization less than or equal to $U_{sum} \leq U_{\mathcal{A}}$, are schedulable by the scheduling algorithm $\mathcal{A}$.*

For example, the utilization bound for RM scheduling [12] on a single core processor is equal to $n \times (2^{\frac{1}{n}} - 1)$, where n is the number of tasks. This bound converges to $\ln(2) \approx 0.69$ when $n \to \infty$. This means that any periodic task set with implicit deadlines that have a total utilization $U_{sum} \leq 0.69$ is schedulable by RM on a single core processor.

Definition 2.2. *The **resource augmentation bound** of a scheduling algorithm $\mathcal{A}$ is defined as the minimum speedup factor b such that: if a task set τ is feasible on m unit-speed cores (i.e. schedulable by an optimal scheduling algorithm), then τ is schedulable by scheduling algorithm $\mathcal{A}$ on m cores that have a speed greater than or equal to b.*

Schedulability Analysis and Schedulability Test

Real-time scheduling algorithms are used to assign jobs to cores at each time instant. In general, they cannot determine whether a task set is schedulable or not. Therefore, **schedulability analyses** are defined and applied at design time to determine if task set τ is schedulable or not by a given scheduling algorithm $\mathcal{A}$. A schedulability analysis corresponding to a scheduling algorithm $\mathcal{A}$, could be based on one of the following approaches:

- **Analytical approach:** it starts by identifying the worst-case scenario of execution (i.e. critical instant) like synchronous arrivals for periodic task sets with constrained deadlines. Then, it determines an analytical schedulability test corresponding to the worst-case scenario identified. This analytical test could be based on a threshold of the utilization factor or density, fixed point of the demand bound function [15] or Response Time Analysis (RTA).
- **Model-checking approach:** it consists in enumerating all reachable states by the system that is composed by the task set τ and the scheduling algorithm $\mathcal{A}$. Then, it analyzes each of these states [16]. This approach have a high complexity and faces a combinatorial state explosion problem but it could be scalable for some cases by using some specific techniques.

- **Simulation-based approach:** it applies the corresponding scheduling algorithm $\mathcal{A}$ on the task set τ in order to build the schedule over a study interval. Then, it checks if all deadlines in this interval are met. In general, a task set releases an infinite sequence of jobs, which means that the study interval should be arbitrarily large to ensure schedulability of τ. However, the study interval could be reduced in some cases while guaranteeing schedulability [17, 18]. For instance, the scheduling scheme of a synchronous and periodic task set is repeated after each hyperperiod.

There are three types of **schedulability test** corresponding to a scheduling algorithm $\mathcal{A}$ described as follows:

- **Sufficient test:** if the test is passed then the task set τ is schedulable by the algorithm $\mathcal{A}$. Otherwise, we cannot conclude: τ can be schedulable or not.
- **Necessary test:** if the test fails then the task set τ is non-schedulable by the algorithm $\mathcal{A}$. Otherwise, we cannot conclude: τ can be schedulable or not.
- **Exact test:** it is both sufficient and necessary schedulability test. Hence, it classifies correctly the task set as schedulable or not.

Remark. *Often, identifying the exact worst-case scenario in order to derive an analytical schedulability test is a difficult problem. Therefore, we add some assumptions to simplify it, but this introduces some pessimism and makes the schedulability test only sufficient and not exact.*

Sustainability and Scheduling Anomalies

A Schedulability analysis allows to determine if a task set is schedulable with a given algorithm. This analysis is done at the design time with theoretical timing parameters such that the worst-case execution time and the minimum inter-arrival time. However, in practice, a real-time system may have slightly different parameter values because tasks do not always run for their whole WCETs and sporadic arrivals occur at the maximum frequency rarely.

Therefore, we should ensure that the deployed system remains schedulable if some parameter changes that reduce the workload happen. This property is known as **sustainability** [19] and it is defined as follows:

Definition 2.3. *A scheduling algorithm is said to be sustainable if a schedulable task set remains schedulable when some of its timing parameters are modified by (i) decreasing execution times, (ii) increasing periods or inter-arrival times, and (iii) increasing deadlines.*

Similarly, a schedulability test is referred to as sustainable if these changes cannot result in a non-schedulable task set that was previously deemed schedulable by the test.

In case of non-sustainable scheduling algorithm, a **scheduling anomaly** occurs when a modification of timing parameters (described above) results in a counter-intuitive effect on schedulability.

2.2 Multi-core Scheduling

Due to an important number of cell phones sells, the silicon vendors have followed this industry by dedicating their development effort to the proposition of multi-core processors or boards. Indeed, after reaching the physical limitation of clock speed due to power consumption and excessive heat dissipation, silicon vendors moved to multi-core processors in order to increase processor performance. Multi-core processors offer a speedup for executing programs by allowing parallel execution of several programs at the same time. This speedup is highly needed in the design of many embedded systems like cell phone industry as it ensures an improved average execution time. However, taking advantage of these performances makes more difficult the worst-case execution time problem by increasing the variability of the execution times. Moreover, more cores requires more complex scheduling algorithms and associated schedulability analyses. Indeed, for a multi-core processor, the scheduling problem consists not only on choosing which task to execute at any given point in time, but also on which core to execute it. Last, but not least, the real-time designers still face difficult problems left open from the single core processors case like taking into account hardware accelerators like pipelines or branch predictors.

Since we present results on homogeneous cores, we restrict our presentation of state of the art to this category of cores. An interested reader may find similar state of the art results for heterogeneous and uniform cores in [14]. We understand here by homogeneous cores, cores that are identical, thus they execute the tasks according to the same rate of execution. The heterogeneous cores are all different, the rate of executing a task is defined for each pair (task, core). The uniform cores have their rate of execution defined with respect to a speed of the core.

Regarding the mapping between tasks and cores, two main categories of scheduling algorithms [20] are defined: the global scheduling and partitioned scheduling policies. A global scheduling policy allows jobs of any task to be scheduled on any core and they may migrate from one core to another core during their execution, if the scheduling policy is preemptive. A partitioned scheduling policy imposes to a task to have all its jobs scheduled on the same core. A third category known as hybrid scheduling integrates properties from both global and partitioned categories.

2.2.1 Global Scheduling

Multi-core processors real-time scheduling theory has it origins in the late 1960's and early 1970's. The seminal paper describing the Dhall effect [21] has an important impact on the research on global scheduling policies during the 1980's and 1990's, as the migration is suffering from such effect. Finally, in 1997, the Dhall effect is proved to be associated to a problem of heavy tasks (important utilization of a core), more than to the migration problem and the real-time community has a regain of interest for the global scheduling policies [22]. This regain allows the appearance of a new thread of results with respect to the optimality of global scheduling algorithms like [23] where Fisher et al. prove that there is no optimal online algorithm for sporadic tasks with constrained or arbitrary deadlines, by showing that such an algorithm would require clairvoyance (i.e. information about future events). Nevertheless, optimal algorithms are proposed. For instance, the Proportionate Fair algorithm allows to each task to execute proportionally to its utilization and this algorithm is proved optimal for periodic tasks with implicit deadlines on identical cores. Different versions of the Proportionate Fair algorithm have been proposed in order to improve its limitations like ERFair which is a work conserving version [24].

Another class of global scheduling policies is related to the introduction of the LLREF algorithm, which is proved optimal for periodic tasks with implicit deadlines on identical cores [25]. The timeline of a schedule is divided in sections and a task is allowed to execute based on the largest local remaining execution time first choice. One noticeable improvement is done for the LRE-TL algorithm that is proved optimal for sporadic tasks with implicit deadlines [26].

Another optimal multiprocessor real-time scheduling algorithm is RUN and it is the first algorithm achieving such optimality without using a fairness or fluid scheduling principle [27], but a dualization technique to transform the multi-core problem in a set of single core problems.

The algorithms presented previously belong to the larger class of job-level priority assignment algorithms. The task-level priority assignment algorithms are, also, well

studied by the real-time community in the context of multi-core processors. Their main detected limitation is presented within the Dhall effect with respect to the lack of the optimality of Rate Monotonic. Answers to these limitation appear as soon as 2001 by showing that any set of implicit deadlines periodic tasks can be scheduled using global RM scheduling if $\max_i(U_i) \leq \frac{m}{3m-2}$ and $U_{sum} \leq \frac{m^2}{3m-1}$ [28], where m is here the number of cores. This condition is improved by showing that the algorithm $RM-US[0.375]$ has 0.375 as maximum utilization bound [29]. More general utilization bounds are later provided by [30].

2.2.2 Partitioned Scheduling

The partitioned scheduling has the main advantage of breaking the scheduling problem into m single core scheduling problems if we consider m identical cores. Most of the time, bin packing techniques, e.g., First-Fit, Best-Fit or Next-Fit, are combined with scheduling algorithms like Rate Monotonic, Deadline Monotonic or EDF that have remarkable optimality properties for given single core processor scheduling problems.

We present here partially the table presented in [14] to underline the advantages and disadvantages of combining such approaches by comparing their approximation ratio.

Algorithm	**Approximation Ratio**
RMBF	2.33
RMFF	2.33
RRM-FF	2
EDF-FF	1.7
EDF-FF	1.7

Table 2.1: Comparison of approximation ratio when algorithms like RM or EDF are combined with FF or BF techniques

The main disadvantage of partitioned scheduling consists in the NP-hardness of deciding the optimal number of cores for a set of tasks. utilization bounds are used to overcome this limitation. For instance, the largest worst-case utilization bound for any partitioning algorithm on a set of periodic implicit deadline tasks is equal $U_{OPT} = \frac{m+1}{2}$ [28], that is later improved by splitting the tasks in two sets according to their utilization that is larger or smaller than $\frac{1}{3}$ [31]. In [23], more general results on partitioned task-level priority assignment policies are provided for constrained and arbitrary deadline periodic tasks.

We present the existing results on the partitioned scheduling of tasks with precedence constraints is Section 2.3.2 that is dedicated to the scheduling of dependent tasks.

2.2.3 Hybrid Scheduling

Hybrid approaches are proposed, recently, to overcome the disadvantages of global and partitioned scheduling algorithms or to answer specific hardware constraints.

Two main classes of algorithms are considered as hybrid: the semi-partitioned approaches and the clustering approaches.

Within the semi-partitioned scheduling algorithms, a limited set of tasks is allowed to migrate while the others are fixed to a core. The most known semi-partitioned scheduling algorithms are EKG [32] with a known utilization bound, EDDP [33], DM-PM [34] and PDMS_HPTS [35]. These algorithms are, mainly proposed by two research groups, indicating limited interest of the community for such approaches.

Within the clustering scheduling algorithms, the cores are grouped in clusters, most of the time those that are the fastest belong to the same cluster. These algorithms have been recently proposed in our community since 2008 by the authors of [36, 37].

2.3 Scheduling of Tasks with Precedence Constraints

2.3.1 Single Core Scheduling

Originally proposed within the operations research community, the scheduling problem of tasks with precedence constraints is considered while a maximum lateness is minimized and the first-to-last-rule is proved optimal [38]. Moreover for the preemptive case Blazewicz gives a polynomial solution [39]. Nevertheless the equivalent real-time problem with different periods and deadlines is a harder problem. For the problem of scheduling dependent periodic tasks on single core processors, Harbour et al. consider a solution based on a definition of a canonical form for the tasks composed of sub-tasks sharing the dependencies [40]. In [41] authors present a first seminal result that is extending a previous paper from the same authors [42] to solve the scheduling problem of sporadic tasks with precedence constraints defined by a direct acyclic graph (DAG). The general idea is the modification of the temporal parameters to ensure the respect of the precedence constraints.

Richard et al.[43] analyze the scheduling problem of graph tasks on single core processor systems where each task consists of a set of dependent fixed-priority sub-tasks with precedence constraints. In order to remove the parallelism within sub-tasks, the authors provided a graph- to-chain transformation that is proved

not to impact the schedulability of the system. Cucu et Sorel consider the non-preemptive version of the same problem as [43] while the schedulability is proved by proving the existence of schedulability intervals [44].

Recently, Stigge et al. [45] consider releases of jobs described by directed graphs and the authors prove that the feasibility problem is decidable in pseudo-polynomial time.

2.3.2 Multi-core scheduling

The scheduling and schedulability problem of real-time systems on multicore processors have been extensively studied [14] after the widespread of these architectures. Besides of sequential task models, the problem of scheduling parallel tasks has been tackled in the literature using different task models.

The fork-join model represents a task as an alternating sequence of sequential and parallel segments. The number of sub-tasks in parallel segments should be the same on all segments and it should not exceed the number of processors. Lakshmanan et al. [46] have proposed a stretch transformation for the fork-join model to execute the parallel segments as sequential when possible.

The synchronous parallel model is considered in [47]. In the latter work, authors present a task decomposition algorithm that transforms implicit deadline tasks into constrained deadline tasks. They also provide a resource augmentation bound for G-EDF and partitioned DM scheduling. This model removes some restrictions of the fork-join model. It allows different numbers of sub-tasks in each segment and these numbers could be greater than the number of processors. However, the synchronization is still required after each parallel segment.

In addition, partitioned scheduling of dependent tasks is studied in the context of distributed systems. Tindell and Clark [48] propose an end-to-end RTA (also known as holistic schedulability analysis) of several independent tasks each composed of a chain of sub-tasks instead of a DAG. This holistic approach was refined later by Palencia et al. [1]. It is used in the MAST tool [49] to analyze multi-path end-to-end flows. This approach is pessimistic since it assumes that higher-priority tasks are always released at each activation of a sub-task from the chain.

A more general parallel task structure is the DAG task model where each task is represented by a direct acyclic graph. Nodes refer to sub-tasks while edges describe precedence between them. A sub-task becomes ready for execution after the satisfaction of all its precedence constraints. This model was studied in the case of global scheduling [50–53]. Indeed, Qamhieh et al. [51] study the schedulability of DAG task model on multiprocessor platform using global EDF

scheduling. They propose to modify the release time and deadline of each sub-task in order to estimate more accurately the workload of a given job. Then, they use the schedulability test proposed in [54]. Fonseca et al. [52] estimate the response time of sporadic DAG tasks under global and fixed-priority scheduling. They use nested fork-join structured DAGs to propose both accurate and efficient solution. In addition, He et al. [53] study the global scheduling of multiple DAG tasks on multi-core processors. They also define sub-tasks execution order inside the same graph to reduce the response time.

DAG task model is also explored under partitioned scheduling system. Fonseca et al. [55] study the response time for sub-tasks scheduled on identical processors according to a partitioned policy. They use self-suspending task [56] to model a DAG task. Then, they estimate the response time of each task by resolving a Mixed Integer Linear Problem (MILP) problem. This approach provides a good estimation of response time compared to existing works but it is not scalable for a relatively large number of tasks because of the complexity of the MILP to solve. Casini et al. [57] focus their work on partitioned, fixed-priority and non-preemptive scheduling of parallel tasks. They propose an approach similar to [55] based on response time analysis of self-suspending tasks [56].

Rihani et al. [58] also study partitioned DAG scheduling on multi-core processors. They suggest to operate offline on a single DAG of multi-rate tasks to generate scheduling table. They also propose to extend their approach to multiple DAG with different period each by unfolding the execution to the hyper-period (the least common multiple of the tasks' periods) which makes the analysis more complex Indeed, the hyperperiod could be potentially large which may explode the number of nodes and increase the complexity. Recently, in [59] the scheduling problem of DAG tasks is formulated as an Integer Linear Program (ILP), when the processor assignments are specified. Hence, an optimal scheduling could be found but it solving a such problem remains NP-hard [60] with high complexity.

2.4 Probabilistic Scheduling

2.4.1 Estimation of Probabilistic Real-time Parameters

The estimation of distributions of probabilistic parameters (pWCET, pWCCT) is tackled in the literature using different techniques. These techniques are divided mainly into two categories; Static Probabilistic Timing Analysis (SPTA) and Measurement-Based Probabilistic Timing Analysis (MBPTA).

Static Probabilistic Timing Analysis (SPTA): In order to build an upper bound of the pWCET distribution of a program, SPTA methods do not execute the program on the target hardware. In fact, they analyze the code and information about input values, along with an abstract model of the hardware behavior. First, they derive information about feasible paths and loop bound by analyzing the code and possible input values. Then, they upper bound the execution time distribution for every path by considering the behavior of hardware features such as pipelines and caches. Finally, they combine these execution time distributions to derive an upper bound on the pWCET distribution of the entire program.

SPTA methods do not explore explicitly different valid scenarios of operation (sequence of input states and hardware states), instead they consider that any scenario could occur. Hence, they yield a valid upper bound on the pWCET distribution for a future scenario of operation. However, this upper bound is obtained after several over-approximations of dynamic behaviors of the program that cannot be precisely determined due to issues of tractability (e.g. cache states in a random replacement cache). Consequently, the estimated pWCET may introduce a significant pessimism.

Measurement-Based Probabilistic Timing Analysis (MBPTA): On the other hand, MBPTA methods execute the program multiple times, on the target hardware, according to several scenarios of operation (i.e. according to a set of feasible input states and initial hardware states) and they measure its execution times under each of these scenarios. Then, they use Extreme Value Theory (EVT) to make a statistical estimate of the pWCET distribution of the program by evaluating the extreme value distribution of measured execution times (samples).

In the literature, there are two main methods to estimate the extreme value distribution [61, 62]; The *Block Maxima* method is based on the Fisher-Tippett-Gnedenko theorem. It splits samples into blocks and selects the maximum value inside each block. These values are used to fit a Generalized Extreme Value (GEV) distribution that estimates the pWCET of the program. Alternatively, the *Peaks-over-Threshold* (PoT) method, which is based on Pickands-Balkema-de Haan theorem, selects values that exceed a suitable threshold. Then, using these values, it fits a Generalized Pareto Distribution (GPD) that estimates the pWCET.

We note that MBPTA methods have lower complexity than SPTA ones when computing an estimate of pWCET since they do not consider all possible paths of the program and all possible inputs and hardware states. However, to guarantee a valid pWCET estimate by MBPTA methods, the chosen scenarios of operation, when

measuring execution time, must be representative of those that will occur during the lifetime of the system. Otherwise, the derived pWCET could under-estimate the actual one and then it could be not a safe upper bound.

Hybrid Probabilistic Timing Analysis (HyPTA): According to the criticality of the application and the available means, one of these two presented methods or both (SPTA and MBPTA) are used to estimate the pWCET. Another category of techniques, called Hybrid Probabilistic Timing Analysis (HyPTA), are used to estimate pWCET. This category combines elements of both static and measurement-based analyses. It may take measurements at the level of sub-paths and then it uses pWCETs of different sub-paths to derive the pWCET of the program by using structural information obtained from static analysis of the code.

Similarly to pWCET estimation, MBPTA techniques are used to estimate the pWCCT distribution of communication time between two programs (sub-tasks). In addition, a similar approach to static analysis is used to estimate communication time via the LIN protocol between ECUs on a vehicle in the work of Byhlin et al. [63].

In [64], the authors statistically estimate the WCET regarding to a given confidence level. This estimate assumes that the collected traces of execution time are independent and it uses the extreme values theory (EVT) in order to approach the distribution of the execution times measured by the Gumbel distribution [65]. The conditioning of this probability by a threshold gives the distribution of WCET. From this distribution, the authors manage to give the level of confidence for each estimate. In addition, they try to determine the WCET estimate at a given confidence level and answer the question if the system is schedulable at that level of confidence.

Other improvements are proposed later, as in [66] which pre-process the collected data, by maximizing the blocks, before applying EVT. Other works have used EVT theory with some improvement on this problem [67–69].

An other approach used for estimating the probabilistic WCET by composition is presented by Bernat et al. in [3]. They use copulas to describe dependencies between different random variables representing WCET of different program. Then they use copulas to compose these programs and compute the overall pWCET and compare it to convolution results.

2.4.2 Single core scheduling

The real-time queueing theory: In [70], *Lehoczky* presents an analysis based on queueing theory. This analysis operates on tasks with an arrival that follows a Poisson distribution of parameter λ and an execution time that follows an exponential law of means $1/\mu$. Actually, these hypotheses are restrictive, but at this time, the paper was one of the first that propose an analysis for a system with probabilistic arrivals and execution times. The author tries to characterize the remaining margin (lead-time) for each task. The problem has a theoretical solution in the case of dense arrivals. This estimate follows the same shape of the empirical distribution of the margin in different cases but it presents a significant difference.

Probabilistic guarantee of schedulability: In the work of *Tia and others* [71], a performance analysis for a set of semi-periodic tasks is proposed. This model describes tasks with a periodic arrivals but with a variable execution time described by a probability distribution. This analysis provides a probabilistic guarantee by two methods.

The first method is called Probabilistic time demand analysis (**PTDA**) and it calculates the probability of meeting the deadline by any instance of a T_i task. This method is based on the classical demand time analysis and it finds a bound on the computation time required by the task T_i and all higher priority tasks. Then, authors estimate the probability that the bound at time t is less than t before the deadline D_i. This probability is derived from the Cumulative Distribution Function (CDF) of the studied bound. Thus, an algorithm is implemented to determine this distribution. Indeed, the algorithm proceeds with a convolution if the number of execution time distributions to sum is less than 10. Otherwise, it uses the central limit theorem to estimate the sum of the probabilistic execution times.

The second method consists on transforming semi-periodic tasks into periodic tasks with constant execution times and sporadic tasks with the remaining execution time (if any). Then, two scheduling approaches are used. The first is to schedule the periodic tasks by the algorithm **RM** [12] while the other tasks are scheduled by a **sporadic server**. An analytic calculation, based on generating functions, is used to evaluate the probability that a sporadic task misses its deadline.

The second approach schedules the periodic tasks by **EDF** algorithm [12] while the sporadic tasks are scheduled by the **Slack Stealing** [72] algorithm. This approach is similar to the one used by *Chetto and Chetto* [42] to schedule periodic tasks with sporadic others by EDF. However, this approach has a linear complexity compared to a polynomial complexity for the solution proposed in [42].

The PTDA analysis proposed in this paper is based on the calculation of a bound of the required computation time, so it is more pessimistic than other recent works. One of these works is *Diaz* work [73] which is discussed later in this section.

Probabilistic execution time: In the paper [73], the authors propose an analysis to calculate the distribution of response time for a system of tasks with a probabilistic execution time. The scheduling used may have a fixed-priority at the task level (like Rate Monotic - RM) or at the job level (like EDF).

First, the authors try evaluate the cost of interruptions using several operations such as convolution, shrink and split. Thus, they evaluate the response time by assuming that the initial backlog of each hyper-period is known.

Second, the authors show that the distribution of the initial backlog follows a Markov chain that is stable when the average utilization rate is less than 1 and even if the maximum utilization exceeds 1. Although the size of Markov transition matrix is infinite, it has a repeating structure. Subsequently, an exact solution is proposed. This solution is expensive in terms of computation, so the authors propose approximations based on iterative calculus or matrix truncation. Finally, they show that the approximate solution to a reasonable order is very close to the exact solution.

Probabilistic inter-arrival time and execution time: In the same context, *Maxim and Cucu* [74] propose a method for calculating response time for probabilistic real-time systems with fixed-priorities. The used model allows tasks to have probabilistic minimum inter-arrival time and execution time. This analysis is based on the convolution and coalescence operations. In addition, authors propose a technique to improve the performance of their analysis. Indeed, after some iterations of the algorithm the size of the distributions explodes because of the successive convolutions. Thus, a resampling technique is proposed to reduce the size of distributions while keeping the distribution shape very close to the original one.

The proposed response time calculation method can be used to construct an optimal fixed-priority probabilistic scheduling algorithm. Indeed, by using the principle of Audsley [75, 76] and this analysis one can assign priorities to these tasks in an optimal way. That means, if the system is feasible then it will meet its deadlines if it is ordered according to these assigned priorities. But this approach is still expensive in terms of computation despite the improvements made by the resampling technique.

Probabilistic execution time with precedence constraints: In [77] we consider multiple values for execution times through probabilistic descriptions and it is dedicated to uniprocessor EDF schedulability of DAGs.

2.4.3 Multi-core scheduling

Probabilistic execution time with precedence constraints: had two main periods of development. The first one is related to the early 2000's regain of interest for probabilistic and statistical approaches, answering an increased pressure from the industrial partners to propose more complex models. During this period, one thread of results is dedicated to the scheduling analysis of DAG task models [78, 79], where a stochastic modelling for the execution times is proposed.

This thesis belongs to the second period started after 2010. Federated scheduling for DAG task model with probabilistic execution time [80] is another such recent work. The difficulties in advancing towards a more general solution does not come from a lack of interest from the real-time community. The difficulties are explained by the fact that the DAG task model is placing the scheduling closer to the modeling stage within the design of a real-time system, while expressing an increased need of describing the functional constraints between tasks.

3

Deterministic DAG Tasks Schedulability on a Multi-core Processor

Contents

3.1 Task Model

We consider a real-time system of n sporadic tasks scheduled according to a partitioned, fixed-priority and preemptive scheduling policy on m identical cores. We denote by τ the set of n tasks $\tau_1, \tau_2, \ldots, \tau_n$ and by π the processor that has m identical and unit-speed cores $\pi_1, \pi_2, \ldots, \pi_m$. Each task τ_i is specified by a 3-tuple (G_i, D_i, T_i), where G_i is a directed acyclic graph (DAG) describing the internal structure of τ_i, D_i is its deadline and T_i the minimum inter-arrival time between two consecutive arrivals. The task τ_i is releasing an infinite sequence of "jobs" separated by at least T_i time units. In the real-time literature these jobs are also known as instances and in the remainder of this thesis we use the term "jobs" to refer to instances. Every job released by τ_i must complete its execution before D_i time units from its release time otherwise we assume that the job is dropped. In

the real-time literature, authors may also consider that these jobs are kept and a backlog is calculated. In this thesis we consider that all jobs that miss their deadline are dropped, thus, there is no backlog accumulation. We consider also constrained deadline tasks (i.e. $D_i \leq T_i,\ \forall i \in \{1, 2, \ldots, n\}$). Thus, any two jobs of the same task cannot be executed at the same time on different cores.

For a task τ_i, the associated DAG G_i is defined by (V_i, E_i), where $V_i = \{\tau_{i,j}\}_{1 \leq j \leq n_i}$ is a set of n_i sub-tasks (vertices) of τ_i. Each sub-task $\tau_{i,j} \in V_i$ represents a computational unit that must be executed sequentially. $E_i \subseteq (V_i \times V_i)$ is the set of the precedence constraints (edges) between the sub-tasks. These sub-tasks could be executed concurrently if they are not related by precedence constraints. However, each sub-task represents a computational unit that must be executed sequentially. A sub-task $\tau_{i,j}$ is characterized by $(C_{i,j}, D_i, T_i)$, where $C_{i,j}$ is its Worst-Case Execution Time (WCET), while D_i and T_i are respectively the deadline and minimum inter-arrival time of $\tau_{i,j}$, which are the same as for the DAG task τ_i.

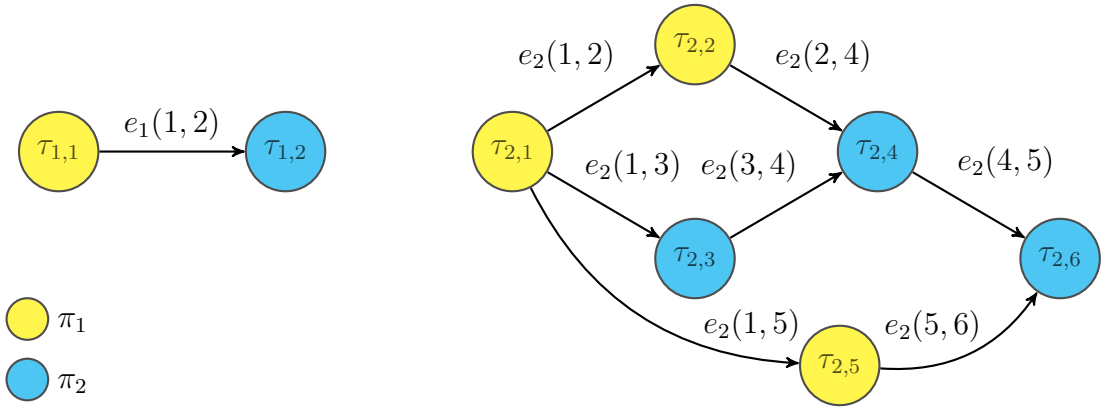


Figure 3.1: Example of DAG tasks describing partitioning and precedence constraints between sub-tasks.

Each sub-task $\tau_{i,j}$ is assigned to only one core and all jobs of that sub-task are scheduled on the same core denoted $\pi(\tau_{i,j})$. For instance, in Figure 3.1, the sub-tasks colored in the same color are scheduled on the same core. Thus, $\tau_{1,1}$, $\tau_{2,1}$, $\tau_{2,2}$ and $\tau_{2,5}$ are scheduled on core π_1, while the other sub-tasks are scheduled on core π_2.

For a task τ_p with a higher priority than a task τ_i, all sub-tasks from τ_p have a higher priority than all sub-tasks of τ_i. For the sake of simplicity, if $p < i$, then τ_p has a higher priority than τ_i. We denote by $hep(\tau_{i,j})$ the set of sub-tasks $\tau_{p,q}$ belonging to **other** DAGs with higher or equal priority to τ_i. More formally, we have $hep(\tau_{i,j}) = \{\tau_{p,q} \in V_p \mid \forall p < i,\ \forall q \in \{1, 2, \ldots, n_p\}\}$. For instance, in Figure 3.1, $hep(\tau_{2,3}) = \{\tau_{1,1}, \tau_{1,2}\}$. For sub-tasks belonging to the **same** DAG as sub-task $\tau_{i,j}$, we consider that they could all have the same priority or each has an individual

priority. Thus, we also define $hep_i(\tau_{i,j})$ as the set of sub-tasks $\tau_{i,k}$ belonging to DAG task τ_i and that have higher or equal priority to $\tau_{i,j}$.

Each directed edge $(\tau_{i,k}, \tau_{i,j}) \in E_i$ denotes a precedence constraint between sub-tasks $\tau_{i,k}$ and $\tau_{i,j}$, meaning that sub-task $\tau_{i,j}$ cannot start executing until sub-task $\tau_{i,k}$ has completed its execution. In this case, $\tau_{i,j}$ is called a "successor" of $\tau_{i,k}$, whereas $\tau_{i,k}$ is called a "predecessor" of $\tau_{i,j}$. Since a sub-task $\tau_{i,j}$ could have multiple predecessors and successors, $\tau_{i,j}$ is said to be "active" if and only if all its predecessors have finished their execution. We call a sub-task without any predecessors or successors, respectively, "source" or "sink" sub-task. A direct acyclic graph could have multiple source and sink sub-tasks. For the sake of simplicity, when computing the response time of the whole DAG, we assume that it has a single sink sub-task. Whenever this assumption does not hold, we add an extra sink sub-task, with an execution time equal to zero, that gathers all sink sub-tasks.

Remark. *We distinguish between "release" and "activation" of a sub-task. When a DAG task is released all its sub-tasks are also released but they are not activated unless all their predecessors have finished their executions.*

Definition 3.1. *We denote the set of the immediate successors of sub-task* $\tau_{i,j}$ *by* $isucc(\tau_{i,j}) = \{\tau_{i,k} \mid \exists\ (\tau_{i,j}, \tau_{i,k}) \in E_i\}$. *We also denote the set of the sub-tasks reachable from* $\tau_{i,j}$ *by directed paths:*

$$succ(\tau_{i,j}) = \{\tau_{i,k} \mid \exists \text{ at least one path from } \tau_{i,j} \text{ to } \tau_{i,k}\}$$

We also define $succ^*(\tau_{i,j})$ *as the set of all successors of sub-task* $\tau_{i,j}$ *including itself, i.e.* $succ^*(\tau_{i,j}) = succ(\tau_{i,j}) \cup \{\tau_{i,j}\}$. *We note that* $isucc(\tau_{i,j}) \subseteq succ(\tau_{i,j}) \subset succ^*(\tau_{i,j})$.

Similarly, we denote the set of immediate predecessors of sub-task $\tau_{i,j}$ by $ipred(\tau_{i,j}) = \{\tau_{i,k} \mid \exists(\tau_{i,k}, \tau_{i,j}) \in E_i\}$. The set of all predecessors of $\tau_{i,j}$ is $pred(\tau_{i,j}) = \{\tau_{i,k} \mid \tau_{i,j} \in succ(\tau_{i,k})\}$. We also define $pred^\star(\tau_{i,j}) = pred(\tau_{i,j}) \cup \{\tau_{i,j}\}$ and we note that $ipred(\tau_{i,j}) \subseteq pred(\tau_{i,j}) \subset pred^\star(\tau_{i,j})$. For instance, in Figure 3.1, immediate predecessors of $\tau_{2,4}$ are $ipred(\tau_{2,4}) = \{\tau_{2,2}, \tau_{2,3}\}$ while $pred(\tau_{2,4}) = \{\tau_{2,1}, \tau_{2,2}, \tau_{2,3}\}$ and $pred^\star(\tau_{1,4}) = \{\tau_{2,1}, \tau_{2,2}, \tau_{2,3}, \tau_{2,4}\}$.

Definition 3.2. *If two sub-tasks in the same graph are not reachable one from the other, they are called independent and they may be executed concurrently whenever they are mapped to different cores. We denote by* $parallel(\tau_{i,j})$ *the set of sub-tasks independent of sub-task* $\tau_{i,j}$. *More precisely,*

$$parallel(\tau_{i,j}) = \{\tau_{i,k} \mid \tau_{i,k} \notin pred^\star(\tau_{i,j}) \cup succ^*(\tau_{i,j})\}$$

For example, in Figure 3.1, sub-tasks $\tau_{2,4}$ and $\tau_{2,5}$ are parallel and they could be executed concurrently since they are assigned to two different cores. We note that $parallel(\tau_{2,5}) = \{\tau_{2,2}, \tau_{2,3}, \tau_{2,4}\}$ but $\tau_{2,2}$ and $\tau_{2,5}$ cannot be executed concurrently because they are mapped to the same core.

A weight $e_i(k, j)$ is associated to each precedence constraint $(\tau_{i,k}, \tau_{i,j}) \in E_i, \forall i \in \{1, 2, \ldots, n\}$. This weight accounts for communication costs between $\tau_{i,k}$ and $\tau_{i,j}$ and it is described by a Worst-Case Communication Time (WCCT). The communication cost is included in the RTA when the sub-tasks are mapped to different cores $(\pi(\tau_{i,k}) \neq \pi(\tau_{i,j}))$. Otherwise, if sub-tasks run on the same core, we assume that the communication delay is reduced and it is included in the WCET of each sub-task. Thus, the communication cost becomes equal to zero. For instance, in Figure 3.1, the communication cost $e_2(3, 4) = 0$ because it relates sub-tasks $\tau_{2,3}$ and $\tau_{2,4}$ that are mapped to the same core, while $e_2(1, 3) \geq 0$ since $\tau_{2,1}$ and $\tau_{2,3}$ run on different cores.

3.2 Deterministic Response Time Analysis

In this section, we propose a response time analysis of the DAG task model defined previously. Our RTA reduces pessimism when estimating Worst-Case Response Time (WCRT) without increasing computational complexity. Indeed, Palencia et al. [1] use fixed point equations to derive an estimation of WCRT. These equations are based on a pessimistic assumption about worst-case arrival patterns that simplifies the analysis but causes an over-estimation of WCRT. On the other hand, Fonseca et al. [55] propose a response time analysis based on solving a MILP optimization problem. Their solution outperforms the work of Palencia et al. [1] and it reduces pessimism in the WCRT estimate. However, it increases the computational complexity and the run-time.

Our proposed response time analysis is based on iterative equations that reduce pessimism compared to Palencia et al. [1]. They reduce also the complexity of the method compared to Fonseca et al. work [55]. In order to present our response time equations, we start by explaining the equations proposed by Palencia et al. in [1]. Then, we describe how we derive our analysis by modifying these equations.

3.2.1 Holistic analysis for sub-task chains on distributed systems

Palencia et al. [1] compute an upper-bound on the response time of a chain of sub-tasks executed on a distributed system (several processors). They proceed sub-task by sub-task from the beginning of the chain. First, they compute the

local response time, denoted $w_{i,j}$, of the studied sub-task $\tau_{i,j}$ when executed on its processor with all higher priority sub-tasks. They assume that these higher priority sub-tasks are activated synchronously with $\tau_{i,j}$ (at the critical instant). Then, they add to the local response $w_{i,j}$, the global response time of the predecessor sub-task of $\tau_{i,j}$ and the communication delay between the two sub-tasks in order to obtain the global response time $R_{i,j}^{global}$ of sub-task $\tau_{i,j}$.

In Equation 3.1, we compute recursively the local response time. It updates monotonically $w_{i,j}$ until reaching a fixed point. This equation is guaranteed to converge if the total utilization of the task set is less than system resources ($U \leq m$).

$$w_{i,j}^{(n+1)} = C_{i,j} + \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{J_{p,q} + w_{i,j}^{(n)}}{T_p} \right\rceil C_{p,q} \tag{3.1}$$

We denote by $J_{p,q}$ the release jitter of sub-task $\tau_{p,q}$ in the sub-task chain (task) τ_p. This jitter represents the variation of the release time of $\tau_{p,q}$ caused by the variation of response time of the predecessor of sub-task $\tau_{p,q}$. Thus, the jitter is equal to the difference between the worst-case and the best-case global response times of the predecessor sub-task. We assume that the best-case response time is equal to zero because it could be arbitrarily small when predecessor sub-tasks do not execute for their entire budget (WCET). Then, we consider that the release jitter is equal to the sum of the worst-case global response time of the predecessor and the communication time between the studied sub-task and its predecessor (see Equation 3.2).

$$J_{i,j} = \begin{cases} 0 & \text{if } \ ipred(\tau_{i,j}) = \varnothing \\ R_{i,k}^{global} + e_i(k,j) & \text{otherwise } (ipred(\tau_{i,j}) = \tau_{i,k}) \end{cases} \tag{3.2}$$

We note that the jitter of the first sub-task (source sub-task) in a sub-task chain is equal to zero because it has no predecessor and no variation in release time. We also notice that a sub-task $\tau_{i,j}$ in the chain has only one immediate predecessor $ipred(\tau_{i,j})$ if it is not a source sub-task.

Remark. *The task model studied by Palencia et al. [1] assumes that individual priority is defined for each sub-task in the same sub-task chain. Nevertheless, we could still use such an analysis on our task model by assigning the same priority to all sub-tasks in the same task.*

The global response time of sub-task $\tau_{i,j}$ (Equation 3.3) is the sum of its worst-case release time and its local response time $w_{i,j}$ (Equation 3.1). The worst-case

release time of sub-task $\tau_{i,j}$ is equal to the response time of the predecessor sub-task ($\tau_{i,k} = ipred(\tau_{i,j})$) with the communication delay $e_i(k,j)$.

$$R_{i,j}^{global} = R_{i,k}^{global} + e_i(k,j) + w_{i,j} \tag{3.3}$$

The global response time of a task τ_i is equal to the global response time of the last sub-task in the chain (the sink sub-task):

$$R_i^{global} = R_{i,sink}^{global} \tag{3.4}$$

Example 3.1. *Through the following task set example (Figure 3.2 and Table 3.1), we explain the importance of including the jitter of preempting sub-tasks in the local response time $w_{i,j}$. If we remove jitter term $J_{p,q}$ from local response time (Equation 3.1) it becomes equal to $w'_{i,j}$ (Equation 3.5):*

$$w'_{i,j} = C_{i,j} + \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{w'_{i,j}}{T_p} \right\rceil C_{p,q} \tag{3.5}$$

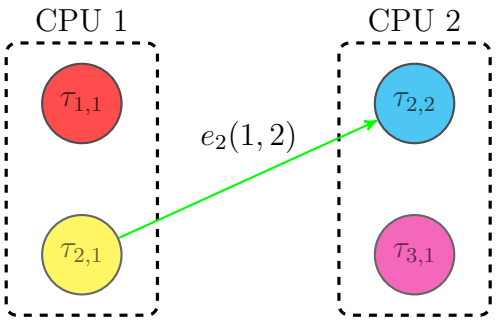


Figure 3.2: Example of a sub-task chains model scheduled on distributed systems and illustrating the importance of jitter

Table 3.1: Parameters of task set example in Figure 3.2

Sub-task	$C_{i,j}$	T_i	Priority
$\tau_{1,1}$	5	10	high
$\tau_{2,1}$	1	15	medium
$\tau_{2,2}$	7	15	medium
$\tau_{3,1}$	4	19	low
$e_2(1,2)$	1	–	–

In Table 3.2, we present the evaluation of the local response time using two formulations, one with the jitter and a second one without the jitter (Equations 3.1

Table 3.2: Evaluating local and global response time of the task set example in Figure 3.2

Sub-task	$\mathbf{w'_{i,j}}$	$\mathbf{R'^{\,global}_{i,j}}$	$\mathbf{J_{i,j}}$	$\mathbf{w_{i,j}}$	$\mathbf{R^{global}_{i,j}}$
$\tau_{1,1}$	5	5	0	5	5
$\tau_{2,1}$	6	6	0	6	6
$\tau_{2,2}$	7	14	7	7	14
$\tau_{3,1}$	11	**11**	0	18	**18**

and 3.5 respectively). We also compute the global response times $R_{i,j}^{global}$ *and* $R'^{\,global}_{i,j}$ *corresponding to each local response time (*$w_{i,j}$ *and* $w'_{i,j}$*).*

We note that the global response time including the jitter $R_{3,1}^{global}$ *is larger than the one that does not include it* $R'^{\,global}_{3,1}$ *for sub-task* $\tau_{3,1}$ *because its preempting sub-task* $\tau_{2,2}$ *has non-zero jitter (*$J_{2,2} = 7$*). In contrast, global response times are equal for other sub-tasks because their preempting sub-tasks have zero jitter. We note also, for sub-task* $\tau_{3,1}$*, that the local response time is equal to its corresponding global response time because it has no predecessor and no release jitter (*$J_{3,1} = 0$*).*

From the scheduling graph in Figure 3.3, we note that the actual response time of the third activation of sub-task $\tau_{3,1}$ *is equal to* $R_{3,1}^{global} = 55 - 38 = 17$*. Unexpectedly, it exceeds the calculated upper bound of the global response time without including jitter* $R'^{\,global}_{3,1} = w'_{3,1} = 11$ *(Table 3.2). Indeed,* $\tau_{3,1}$ *could be preempted by sub-task* $\tau_{2,2}$ *since it has higher priority and is executed on the same processor. Besides,* $\tau_{2,2}$ *could actually have two successive* ***activations*** *separated by* 10 *time units (Figure 3.3) instead of* 15 *its minimum inter-arrival time and this is due to the release jitter. However, in the iterative Equation 3.5 for* $w'_{3,1}$ *computation, the ceiling term is equal to* 1 *for an interval of length* $w'_{3,1} = 11$ *when the fixed point is reached. Then, the activation of* $\tau_{2,2}$ *is considered only once in the response time of the sub-task* $\tau_{3,1}$ *which explains the under-estimation. Hence, excluding the release jitter from the local response time (Equation 3.1) may lead to erroneous and non-safe estimation of the worst-case response time.*

3.2.2 Extension of holistic analysis for a DAG task model

The holistic analysis for distributed systems (Section 3.2.1), implemented through Equations 3.1, 3.2 and 3.3, operates only on tasks composed of a single path without any parallel sub-tasks. Moreover, priorities are defined at task level because at sub-task level there is only one order of execution; it is the one that respects precedence constraints in the sub-task chain. However, in a DAG task model, several orders among sub-tasks could exist because there are parallel sub-tasks and multiple paths within a single task.

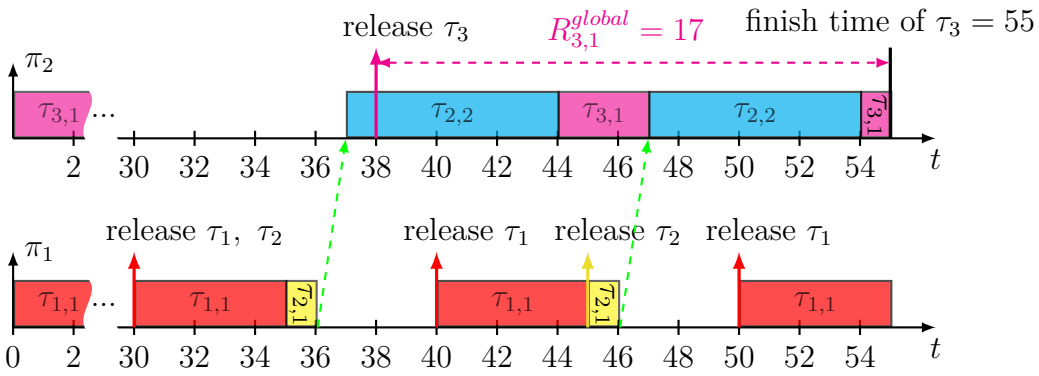


Figure 3.3: The impact of the release jitter of $\tau_{2,2}$ on the scheduling of task set defined in Figure 3.2 and Table 3.1.

In this part, we present three methods to extend holistic analysis from a sub-task chains model (Section 3.2.1) to a DAG task model with parallel sub-tasks. First, we add to the equations analyzing the sub-task chains model, the effect of parallel sub-tasks from the same graph that are executed on the same core. Next, we identify and avoid counting the same sub-task several times because it is parallel to different sub-tasks. Thus, we reduce the pessimism and the over-estimation of the worst-case response time. Moreover, in the proposed extensions, we consider the two cases when priorities are defined at task level and at sub-task level. We provide a generic formulation for all equations by using the set $hep_i(\tau_{i,j})$ of higher or equal priority sub-tasks inside the same graph. Depending on the definition of this set $hep_i(\tau_{i,j})$, the equations are adapted to the desired definition of the level of priority .

In addition, the holistic analysis is applied only on periodic task sets. Thus, the derived extensions also operate on periodic task sets. However, we could use them to study the schedulability of sporadic task sets on a partitioned multi-core processor. Indeed, the multi-core partitioned scheduling problem could be seen as several single core processor scheduling problems once the allocation is done [14]. On the other hand, Baruah and Burns [19] show that response time analysis of fixed-priority and preemptive scheduling on a single core processor incorporating release jitter and blocking time is sustainable with respect to the period. Hence, if a periodic task set is schedulable with minimum inter-arrival time as the period for all tasks then it is schedulable with higher period and thus the sporadic system is schedulable. We conclude that the resulting sporadic systems on each core are all schedulable and so the whole partitioned task set is also schedulable.

3.2.2.1 First method: Including parallel execution in local response time

Unlike the sub-task chains model, the same DAG task may have several parallel sub-tasks that contend to run on the same core at the same time. However, the expression of the local response time $w_{i,j}$ (Equation 3.1) does not consider these parallel computations inside the same DAG task.

Lemma 3.1. *The local response time of sub-task $\tau_{i,j}$ belonging to a DAG task model is given by the following equation:*

$$w_{i,j} = C_{i,j} + \sum_{\tau_{\mathbf{i,k}} \in \mathbf{P_{i,j}}} \mathbf{C_{i,k}} + \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{J_{p,q} + w_{i,j}}{T_p} \right\rceil C_{p,q} \tag{3.6}$$

Proof. In order to obtain the local response time of a sub-task in a DAG task model, we add to the formulation of $w_{i,j}$ (Equation 3.1) the effect of concurrent sub-tasks executed on the same core (bold term in Equation 3.6). Indeed, we upper bound this effect by the sum of execution times of sub-tasks in $\mathbf{P_{i,j}}$.

We denote by $\mathbf{P_{i,j}}$ the set of sub-tasks from the same DAG that could preempt $\tau_{i,j}$. For a sub-task $\tau_{i,k}$ to be able to **preempt** a sub-task $\tau_{i,j}$, it should be:

- Parallel to $\tau_{i,j}$.
- Run on the same core as $\tau_{i,j}$.
- Have higher or equal priority to $\tau_{i,j}$.

More formally, we obtain:

$$\mathbf{P_{i,j}} = \{\tau_{i,k} \in V_i \setminus \{\tau_{i,j}\} \mid \tau_{i,k} \in parallel(\tau_{i,j}),\ \pi(\tau_{i,k}) = \pi(\tau_{i,j}),\ \tau_{i,k} \in hep_i(\tau_{i,j})\}$$

■

Remark. *In the case where priorities are defined at sub-task level, we assume that the set $hep_i(\tau_{i,j})$ is given for all sub-task $\tau_{i,j}$. If priorities are defined only at task level, then the set $hep_i(\tau_{i,j})$ is composed of all sub-tasks in τ_i except $\tau_{i,j}$. The condition $\tau_{i,k} \in hep_i(\tau_{i,l})$ becomes equivalent to $\tau_{i,k} \in V_i \setminus \{\tau_{i,j}\}$. Therefore, the formulation of $P_{i,j}$ could be simplified as below but the previous formulation remains more generic and correct for the two cases of defining the level of priority.*

$$\mathbf{P_{i,j}} = \{\tau_{i,k} \in V_i \setminus \{\tau_{i,j}\} \mid \tau_{i,k} \in parallel(\tau_{i,j}),\ \pi(\tau_{i,k}) = \pi(\tau_{i,j})\}$$

In addition, within the DAG task model, a sub-task could have several immediate predecessors. Since the release jitter is the worst-case response time of predecessor sub-tasks, then it becomes equal to the maximum, over all immediate predecessors, of the global response time added to the corresponding communication delay (Equation 3.7).

$$J_{i,j} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{global} + e_i(k,j) \right\} \tag{3.7}$$

Theorem 3.1. *The global response time of sub-task $\tau_{i,j}$ belonging to a DAG task model is given by the following equation:*

$$R_{i,j}^{global} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{global} + e_i(k,j) \right\} + w_{i,j} \tag{3.8}$$

Proof. The local response time of a sub-task $\tau_{i,j}$ takes into consideration all possible preemptions caused by higher priority DAGs and by parallel sub-tasks on other paths. Then, from its activation, $\tau_{i,j}$ requires in the worst-case its local response time $w_{i,j}$ to finish its execution .

The maximum term in Equation 3.8 includes the worst-case communication delay with the global response time that takes into account the effect of parallel sub-tasks. Thus, this term provides sufficient time for all predecessors even on different paths to be executed. After that, $\tau_{i,j}$ is activated and it could start executing.

Therefore the global response time of $\tau_{i,j}$ is equal to the sum of its activation date (the maximum term) and its local response time $w_{i,j}$.

■

The previous equations, used for calculating an upper-bound of the response time, could be written in a more explicit manner. Therefore, we define the *internal interference* and the *external interference*.

Definition 3.3. *Let the internal interference $I_i^{int}(\tau_{i,j})$ be the maximum cumulative time during which sub-task $\tau_{i,j}$ is active but cannot execute because its assigned core $\pi(\tau_{i,j})$ is executing other sub-tasks belonging to the same DAG task τ_i. This intra-task interference $I_i^{int}(\tau_{i,j})$ is caused by parallel sub-tasks from τ_i on sub-task $\tau_{i,j}$ and it is equal to the sum of parallel sub-tasks that executed on the same core and have a priority that is higher or equal to $\tau_{i,j}$:*

$$I_i^{int}(\tau_{i,j}) = \sum_{\tau_{i,k} \in P_{i,j}} C_{i,k} \tag{3.9}$$

Remark. *We consider that the deadline of a DAG task τ_i is constrained (i.e. $D_i \leq T_i$). We also assume that if a job of τ_i miss its deadline, then it is dropped. Hence, two jobs of the same DAG task τ_i cannot be active at the same time.*

Let $\tau_{i,k}$ be a sub-task that belongs to the DAG task τ_i. We assume that $\tau_{i,k}$ is assigned to the same core and have a higher priority than another sub-task $\tau_{i,j}$. If $\tau_{i,k}$ could preempt $\tau_{i,j}$, then the two sub-tasks should belong to the same job of the DAG task τ_i and they should be parallel to each other. Consequently, $\tau_{i,k}$ could preempt $\tau_{i,j}$ only once during each period and it is considered ***only once*** *in the internal interference $I_i^{int}(\tau_{i,j})$.*

Definition 3.4. *Let the external interference $I^{ext}(\tau_{i,j})$ be the maximum cumulative time during which sub-task $\tau_{i,j}$ is active to execute but cannot because its assigned core $\pi(\tau_{i,j})$ is executing sub-tasks from other DAGs. This inter-task interference is caused by sub-tasks belonging to any higher priority DAG task.*

Lemma 3.2. *The local response time could be expressed using the internal and external interference as follows:*

$$w_{i,j} = C_{i,j} + I_i^{int}(\tau_{i,j}) + I^{ext}(\tau_{i,j}) \tag{3.10}$$

Proof. By definition, in the work of Palencia et al. [1], the local response time $w_{i,j}$ is an upper bound of the time required by sub-task $\tau_{i,j}$ to finish its execution from its activation. Thus, $w_{i,j}$ is equal to the sum of the execution time $C_{i,j}$ and of the delay during which sub-task $\tau_{i,j}$ is active but cannot execute because its assigned core $\pi(\tau_{i,j})$ is executing other sub-tasks. This delay is composed of the internal interference $I_i^{int}(\tau_{i,j})$ and the external interference $I^{ext}(\tau_{i,j})$. ■

The local response time $w_{i,j}$ (Equation 3.6) could be written:

$$w_{i,j} = C_{i,j} + I_i^{int}(\tau_{i,j}) + \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{J_{p,q} + w_{i,j}}{T_p} \right\rceil C_{p,q}$$

$$w_{i,j} - C_{i,j} - I_i^{int}(\tau_{i,j}) = \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{J_{p,q} + w_{i,j}}{T_p} \right\rceil C_{p,q} \qquad \text{(using Equation 3.10)}$$

$$I^{ext}(\tau_{i,j}) = \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{J_{p,q} + I^{ext}(\tau_{i,j}) + C_{i,j} + I_i^{int}(\tau_{i,j})}{T_p} \right\rceil C_{p,q} \tag{3.11}$$

We note that $I^{ext}(\tau_{i,j})$ is expressed recursively using Equation 3.11.

Corollary 3.1. *The global response time of sub-task $\tau_{i,j}$ could be expressed as follows:*

$$R_{i,j}^{global} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{global} + e_i(k,j) \right\} + C_{i,j} + I_i^{int}(\tau_{i,j}) + I^{ext}(\tau_{i,j}) \quad (3.12)$$

Proof. Based on Theorem 3.1 and Lemma 3.2, we conclude that the global response time of sub-task $\tau_{i,j}$ could be written as the sum of the maximum global response time over immediate predecessors with the communication delay, the execution time $C_{i,j}$, the internal and external interference exerted on $\tau_{i,j}$. ■

Since we have assumed that a DAG task has only one sink sub-task, the global response time of a DAG task τ_i is equal to the global response time of its sink sub-task:

$$R_i^{global} = R_{i,sink}^{global} \quad (3.13)$$

Example 3.2. *In this example, we illustrate how the previous equations include parallel computations from the same graph. Thus, we consider two DAG tasks defined in Figure 3.4 and Table 3.3. Task τ_1 has a period of $T_1 = 20$ while $T_2 = 50$. Task τ_1 has higher priority than τ_2 .*

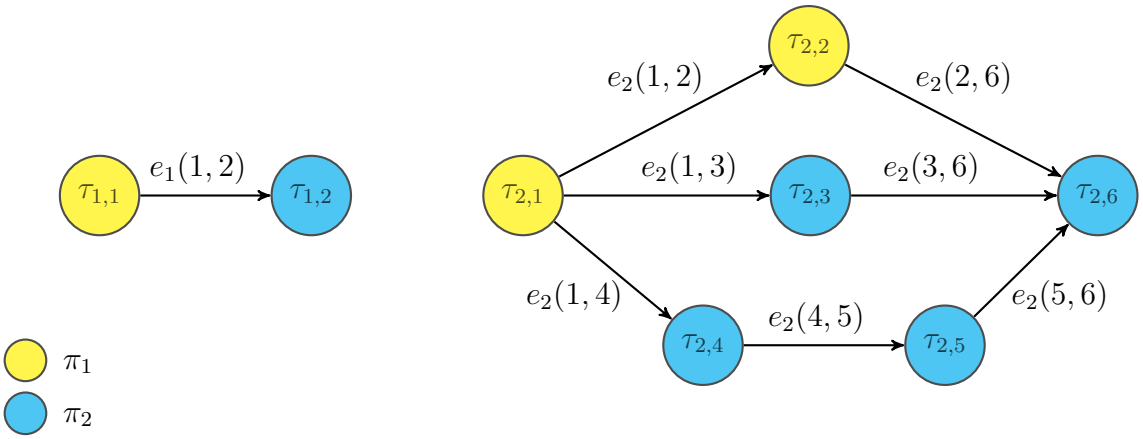


Figure 3.4: Example of a DAG task set with parallel sub-tasks

Table 3.3: Parameters of task set described in Figure 3.4

Sub-task	$C_{i,j}$	T_i	Priority	Precedence	delay
$\tau_{1,1}$	3	20	high	$e_1(1,2)$	1
$\tau_{1,2}$	1	20	high	$e_2(1,2)$	0
$\tau_{2,1}$	2	50	low	$e_2(1,3)$	1
$\tau_{2,2}$	1	50	low	$e_2(1,4)$	1
$\tau_{2,3}$	3	50	low	$e_2(4,5)$	0
$\tau_{2,4}$	1	50	low	$e_2(2,6)$	1
$\tau_{2,5}$	1	50	low	$e_2(3,6)$	0
$\tau_{2,6}$	2	50	low	$e_2(5,6)$	0

In Table 3.4, we show the results obtained by considering parallel execution in the local response time. We use Equations 3.9, 3.11 and 3.12 to compute an upper-bound of the response time of the previous DAG tasks with parallel sub-tasks (Figure 3.4).

Table 3.4: Estimation of the worst-case response time of sub-tasks described in Figure 3.4

Sub-task	$\mathbf{C_{i,j}}$	$\mathbf{w_{i,j}}$	$\mathbf{J_{i,j}}$	$\mathbf{I_i^{int}}(\tau_{\mathbf{i,j}})$	$\mathbf{I^{ext}}(\tau_{\mathbf{i,j}})$	$\mathbf{R_{i,j}^{global}}$
$\tau_{1,1}$	3	3	0	0	0	3
$\tau_{1,2}$	1	1	4	0	0	5
$\tau_{2,1}$	2	5	0	0	3	5
$\tau_{2,2}$	1	4	5	0	3	9
$\tau_{2,3}$	3	6	6	2	1	12
$\tau_{2,4}$	1	5	6	3	1	11
$\tau_{2,5}$	1	5	11	3	1	**16**
$\tau_{2,6}$	2	3	16	0	1	**19**

From Table 3.4, we note that the estimated worst-case response time includes the execution time of parallel sub-tasks. For instance, the local response times $w_{2,4}$ and $w_{2,5}$ of sub-tasks $\tau_{2,4}$ and $\tau_{2,5}$ respectively include the execution time $C_{2,3}$ of their parallel sub-task $\tau_{2,3}$. Similarly, the local response time $w_{2,3}$ includes the execution times $C_{2,4}$ and $C_{2,5}$. However, the global response time $R_{2,5}^{global}$ of sub-task $\tau_{2,5}$ includes the execution time of $\tau_{2,3}$ twice: once from the global response time of its predecessor $R_{2,4}^{global}$ and once from its local response time $w_{2,5}$. In general, if a sub-task $\tau_{i,j}$ is parallel and runs on the same core as many sub-tasks that belong to the same path, then the execution time of $\tau_{i,j}$ will be included several times and propagated through this path. Thus, the response time of the last sub-task in the path will be over-estimated and will suffer from a snowball effect.

We conclude that the proposed response time Equations 3.9, 3.11 and 3.12 take into consideration the effect of parallel sub-tasks but they are very pessimistic. Indeed, these equations may include the same sub-task several times if it is parallel to several related sub-tasks that are executed on the same core. Thus, we propose to use another way to take into account parallel executions without over-estimating the worst-case response time.

3.2.2.2 Second method: Including parallel execution in global response time

In order to avoid including parallel sub-tasks several times when calculating an upper-bound of the response time, we propose to compute, first, a sequential response time.

Definition 3.5. *The sequential response time $R_{i,j}^{seq}$ takes into consideration only the execution time $C_{i,j}$ of the studied sub-task $\tau_{i,j}$, the maximum sequential response time over its immediate predecessors and the external interference but it does not consider the internal interference caused by parallel sub-tasks. The sequential response time is defined as follows:*

$$R_{i,j}^{seq} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{seq} + e_i(k,j) \right\} + C_{i,j} + I^{ext}(\tau_{i,j}) \tag{3.14}$$

Second, we compute the global response time by adding to the sequential response time the effect of parallel sub-tasks on the studied sub-task $\tau_{i,j}$ and on all its predecessors $pred(\tau_{i,j})$.

Theorem 3.2. *The global response time of sub-task $\tau_{i,j}$ is given by the following equation:*

$$R_{i,j}^{global} = R_{i,j}^{seq} + \sum_{\tau_{\mathbf{i,k}} \in \mathbf{\Pi_{i,j}}} \mathbf{C_{i,k}} \tag{3.15}$$

We denote by $\mathbf{\Pi_{i,j}}$ the set of parallel sub-tasks that could preempt one of the sub-tasks in $pred^{\star}(\tau_{i,j})$ containing $\tau_{i,j}$ and all its predecessors.

A sub-task $\tau_{i,k}$ may preempt a sub-task $\tau_{i,l}$, if they are parallel and run on the same core. It should also have a priority that is higher than or equal to $\tau_{i,l}$.

$$\mathbf{\Pi_{i,j}} = \{\tau_{i,k} \in V_i \mid \exists\ \tau_{i,l} \in pred^{\star}(\tau_{i,j}) \text{ such that } \tau_{i,k} \in P_{i,l}\}$$

We note that the set $\Pi_{i,j}$ could contain predecessors of $\tau_{i,j}$ that are parallel to other predecessors of $\tau_{i,j}$ on other paths.

Proof. In an oriented graph, the *longest path* from source nodes to a given node $\tau_{i,j}$ is equivalent to the *downward rank* used in list scheduling [81, 82]. It is computed by considering the maximum, over immediate predecessors, of the sum of the longest path to the predecessor considered, the cost of this predecessor and the weight between the predecessor and the target node.

$$L_{i,j} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \{L_{i,k} + cost(\tau_{i,k}) + weight(k,j)\}$$

Let $L'_{i,j} = L_{i,j} + cost(\tau_{i,j})$ then:

$$L'_{i,j} - cost(\tau_{i,j}) = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \Big\{ L'_{i,k} + weight(k,j) \Big\}$$
$$L'_{i,j} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \Big\{ L'_{i,k} + weight(k,j) \Big\} + cost(\tau_{i,j})$$

In the previous equation, if we replace $weight(k,j)$ by $e_i(k,j)$ and $cost(\tau_{i,j})$ by $C_{i,j} + I^{ext}(\tau_{i,j})$ we obtain a similar equation to Equation 3.14 that defines the sequential response time. Thus, $R^{seq}_{i,j}$ represents the longest path that considers the the individual execution time $C_{i,j}$ and the external interference $I^{ext}(\tau_{i,j})$ but omits the effect of parallel sub-tasks.

In order to compute the global response time, we add to the sequential response time, the internal interference caused by parallel sub-tasks (in the set $\Pi_{i,j}$) to the studied sub-task $\tau_{i,j}$ and its predecessors. To upper bound this interference, we consider the worst-case and we neglect any concurrent execution of sub-tasks in $\Pi_{i,j}$ even if they run on different cores. Therefore, we sum the execution time of all sub-tasks in $\Pi_{i,j}$ to safely estimate their effect.

■

Example 3.3. *We use the previous example defined in Figure 3.4 and Table 3.3 for the purpose of illustrating how the proposed response time Equations 3.14 and 3.15 take into consideration parallel sub-tasks. We also show how these equations allow us to avoid including the effect of the same parallel sub-task several times when browsing the graph and to avoid a snowball effect on the sink sub-task.*

Table 3.5: Applying response time Equations 3.14 and 3.15 on the task set described in Figure 3.4

Sub-task	$\mathbf{C_{i,j}}$	$\mathbf{J_{i,j}}$	$\mathbf{I_i^{int}}(\tau_{i,j})$	$\mathbf{I^{ext}}(\tau_{i,j})$	$\mathbf{R^{seq}_{i,j}}$	$\mathbf{\Pi_{i,j}}$	$\mathbf{R^{global}_{i,j}}$
$\tau_{1,1}$	3	0	0	0	3	$\varnothing$	3
$\tau_{1,2}$	1	4	0	0	5	$\varnothing$	5
$\tau_{2,1}$	2	0	0	3	5	$\varnothing$	5
$\tau_{2,2}$	1	5	0	3	9	$\varnothing$	9
$\tau_{2,3}$	3	6	2	1	10	$\tau_{2,4}, \tau_{2,5}$	12
$\tau_{2,4}$	1	6	3	1	8	$\tau_{2,3}$	11
$\tau_{2,5}$	1	11	3	1	10	$\tau_{2,3}$	**13**
$\tau_{2,6}$	2	13	0	1	13	$\tau_{2,3}, \tau_{2,4}, \tau_{2,5}$	**18**

In Table 3.5, we give the results obtained by including parallel executions in the global response time. We note that the response time of the sub-task $\tau_{2,5}$ *is reduced to* $R^{global}_{2,5} = 13$ *instead of* 16 *(in Table 3.4) and the response time of the*

DAG task τ_2 is reduced from $R_2^{global} = R_{2,6}^{global} = 19$ to 18. Indeed, a parallel sub-task to other sub-tasks from the same path, like $\tau_{2,3}$, is no longer included several times. However, sub-tasks that are included in the sequential response time of the sink sub-task may be computed twice in the global response time if they are parallel to other sub-tasks. For example, the execution times of sub-tasks $\tau_{2,4}$ and $\tau_{2,5}$ are included in the sequential response time $R_{2,6}^{seq}$ (Equation 3.14) through the maximum term over immediate predecessors. Moreover, they are computed a second time in the global response time (Equation 3.15) through the sum term since $\tau_{2,4}$ and $\tau_{2,5}$ are members of the set $\Pi_{2,6}$ (i.e. they are parallel and could preempt $\tau_{2,6}$ or to one of its predecessors).

To summarize, Equations 3.14 and 3.15 take into account the effect of parallel sub-tasks while reducing pessimism and over-estimation of the worst-case response time. They avoid computing several times the same sub-task that is parallel to other sub-tasks belonging to the same path and executing on the same core. Nevertheless, these equations still over-estimate the worst-case response time because some tasks in the critical path to the sink task may be computed twice if they are parallel to other sub-tasks and mapped to the same core. Hence, in the next part, we revise Equations 3.14 and 3.15 to further reduce the pessimism.

3.2.2.3 Third method: Including parallel execution between predecessors

To avoid considering parallel sub-tasks on the critical path twice when computing sequential and global response times, we propose to consider first the effect of parallel execution from different paths that are predecessors to the studied sub-task. Then, we add the effect of parallel sub-tasks that are not predecessors. We denote by $R_{i,j}^{pred}$ the preceding response time that considers the effect of higher priority DAG tasks and the effect of parallel sub-tasks that are predecessors of $\tau_{i,j}$ while it omits the effect of parallel and not predecessor sub-tasks.

Definition 3.6. *Let $\tau_{i,k}$ be an immediate predecessor of a sub-task $\tau_{i,j}$ ($\tau_{i,k} \in ipred(\tau_{i,j})$). We define $\mathbf{I_{i,j}^{pred}}(\tau_{\mathbf{i,k}})$ as the internal interference exerted on $\tau_{i,k}$ and its predecessors by other predecessors of $\tau_{i,j}$.*

Lemma 3.3. *The internal interference $I_{i,j}^{pred}(\tau_{i,k})$ is given by:*

$$I_{i,j}^{pred}(\tau_{i,k}) = \begin{cases} \sum\limits_{\tau_{i,l} \in \Psi_{i,j}(\tau_{i,k})} C_{i,l} & \text{if } \tau_{i,k} \in ipred(\tau_{i,j}) \\ 0 & otherwise \end{cases} \tag{3.16}$$

We denote by $\mathbf{\Psi_{i,j}}(\tau_{\mathbf{i,k}})$ the set of predecessor sub-tasks to $\tau_{i,j}$ but not predecessors to $\tau_{i,k}$ that could preempt one of the sub-tasks in $pred^{\star}(\tau_{i,k})$ containing $\tau_{i,k}$ and all its predecessors. We recall that $\tau_{i,k} \in ipred(\tau_{i,j})$, otherwise the set $\Psi_{i,j}(\tau_{i,k})$ is not defined ($\Psi_{i,j}(\tau_{i,k}) = \varnothing$).

$$\mathbf{\Psi_{i,j}}(\tau_{\mathbf{i,k}}) = \{\tau_{i,l} \in pred(\tau_{i,j}) \setminus pred^{\star}(\tau_{i,k}) \mid \exists \tau_{i,a} \in pred^{\star}(\tau_{i,k}) \text{ such that } \tau_{i,l} \in P_{i,a}\} \tag{3.17}$$

Proof. To estimate the worst-case interference caused by a sub-task $\tau_{i,l} \in \Psi_{i,j}(\tau_{i,k})$, we assume that the execution of $\tau_{i,l}$ on core $\pi(\tau_{i,l})$ delays not only predecessors of $\tau_{i,k}$ that are executed on core $\pi(\tau_{i,l})$ but all sub-tasks $\tau_{i,a} \in pred^{\star}(\tau_{i,k})$ (whether they are executed on $\pi(\tau_{i,l})$ or not). In fact, such sub-task $\tau_{i,a}$ may be a successor of another sub-task in $pred^{\star}(\tau_{i,k})$ that runs on $\pi(\tau_{i,l})$ and is delayed by $\tau_{i,l}$.

Therefore, the worst-case interference that sub-tasks of $\Psi_{i,j}(\tau_{i,k})$ cause on the preceding response time of $\tau_{i,k}$, is equal to the sum of the execution times of all the sub-tasks $\tau_{i,l} \in \Psi_{i,j}(\tau_{i,k})$.

■

Lemma 3.4. *The preceding response time $R_{i,j}^{pred}$ is computed by the following equation:*

$$R_{i,j}^{pred} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{pred} + e_i(k,j) + \mathbf{I_{i,j}^{pred}}(\tau_{\mathbf{i,k}}) \right\} + C_{i,j} + I^{ext}(\tau_{i,j}) \tag{3.18}$$

Proof. To prove Equation 3.18, we use mathematical induction. First, we verify this equation for source sub-tasks. The preceding response time of a sub-task $\tau_{i,j}$ considers $\tau_{i,j}$, its predecessors and higher priority DAGs while discarding the effect of parallel sub-tasks that are not predecessors. If $\tau_{i,j}$ is a source sub-task without any predecessor, then its preceding response time is equal to its execution time with the effect of sub-tasks belonging to higher priority DAGs (external interference). Since there are no predecessor sub-tasks to $\tau_{i,j}$, the "maximum" term, in Equation 3.18 is equal to zero. Therefore, the computed $R_{i,j}^{pred}$ is equal to the execution time $C_{i,j}$ with the external interference and Equation 3.18 is verified for source sub-tasks.

Now, we assume that Equation 3.18 is valid for all predecessors of a sub-task $\tau_{i,j}$ and we prove that Equation 3.18 is correct for $\tau_{i,j}$. Indeed, for each immediate predecessor $\tau_{i,k}$ of $\tau_{i,j}$, we assume that $R_{i,k}^{pred}$ is enough for $\tau_{i,k}$ and all its predecessors to finish their executions. Besides, the maximum interference caused on $\tau_{i,k}$ by other predecessors of $\tau_{i,j}$ is equal to $I_{i,j}^{pred}(\tau_{i,k})$ as explained in Equation 3.16. Since we consider only predecessor sub-tasks in the preceding response time, the latest start time of $\tau_{i,j}$ is equal to the maximum, over immediate predecessors $\tau_{i,k}$, of the sum

of: (i) the preceding response time of $\tau_{i,k}$ with the corresponding communication delay $e_i(k,j)$. (ii) the maximum interference $I_{i,j}^{pred}(\tau_{i,k})$ caused by other predecessors of $\tau_{i,j}$.

The "maximum" term in Equation 3.18 provides sufficient time for all predecessors of $\tau_{i,j}$ to be executed and then $\tau_{i,j}$ starts executing. Since the preceding response time considers higher priority DAGs but not parallel sub-tasks, $\tau_{i,j}$ finishes its execution after $C_{i,j} + I^{ext}(\tau_{i,j})$ from its start. Hence, we add, to the "maximum" term, the execution time $C_{i,j}$ and the external interference $I^{ext}(\tau_{i,j})$ exerted on $\tau_{i,j}$ in order to get the preceding response time $R_{i,j}^{pred}$.

In conclusion, we prove the correctness of Equation 3.18 for any sub-task $\tau_{i,j}$ by applying the previous property for all its predecessors. We start from source sub-tasks and we move step-by-step to successors until reaching $\tau_{i,j}$.

■

Theorem 3.3. *The global response time is computed as follows:*

$$R_{i,j}^{global} = R_{i,j}^{pred} + \sum_{\tau_{\mathbf{i,k}} \in \mathbf{\Pi_{i,j}^{pred}}} \mathbf{C_{i,k}} \tag{3.19}$$

We denote by $\mathbf{\Pi_{i,j}^{pred}}$ *the set of sub-tasks not predecessors to* $\tau_{i,j}$ *that could preempt* $\tau_{i,j}$ *or one of its predecessors.*

$$\mathbf{\Pi_{i,j}^{pred}} = \{\tau_{i,k} \in V_i \setminus pred^{\star}(\tau_{i,j}) \mid \exists\ \tau_{i,l} \in pred^{\star}(\tau_{i,j}) \text{ such that } \tau_{i,k} \in P_{i,l}\} \tag{3.20}$$

Proof. To obtain the global response time, we add to the preceding response time $R_{i,j}^{pred}$, the interference exerted on $\tau_{i,j}$ and its predecessors by parallel sub-tasks that are not predecessors to $\tau_{i,j}$. The parallel sub-tasks considered should not be predecessors to $\tau_{i,j}$ because the effect of parallel execution among predecessors of $\tau_{i,j}$ is already included in the preceding response time $R_{i,j}^{pred}$ (Equation 3.18) by the interference term $I_{i,j}^{pred}(\tau_{i,k})$. Thus, we consider the set $\Pi_{i,j}^{pred}$ containing these parallel sub-tasks (Equation 3.20).

Similarly to the proof of Lemma 3.3, we prove that the maximum internal interference that sub-tasks of $\Pi_{i,j}^{pred}$ cause on $\tau_{i,j}$ and its predecessors, is equal to the sum of the execution times of all sub-tasks $\tau_{i,k} \in \Pi_{i,j}^{pred}$.

■

Example 3.4. *Using the previous example defined in Figure 3.4 and Table 3.3, we illustrate how the proposed response time Equations 3.18 and 3.19 help to reduce pessimism and to avoid including the execution time of sub-tasks on the critical path twice.*

Table 3.6: Applying response time Equations 3.18 and 3.19 on task set described in Figure 3.4

Sub-task	$\mathbf{C_{i,j}}$	$\mathbf{J_{i,j}}$	$\mathbf{I^{ext}}(\tau_{\mathbf{i,j}})$	$\mathbf{R^{pred}_{i,j}}$	$\mathbf{\Pi^{pred}_{i,j}}$	$\mathbf{R^{global}_{i,j}}$	**Edge**	$\mathbf{\Psi_{i,j}}(\tau_{\mathbf{i,k}})$	$\mathbf{I^{pred}_{i,j}}(\tau_{\mathbf{i,k}})$
$\tau_{1,1}$	3	0	0	3	$\varnothing$	3	$e_1(1,2)$	$\varnothing$	0
$\tau_{1,2}$	1	4	0	5	$\varnothing$	5	$e_2(1,2)$	$\varnothing$	0
$\tau_{2,1}$	2	0	3	5	$\varnothing$	5	$e_2(1,3)$	$\varnothing$	0
$\tau_{2,2}$	1	5	3	9	$\varnothing$	9	$e_2(1,4)$	$\varnothing$	0
$\tau_{2,3}$	3	6	1	10	$\tau_{2,4}, \tau_{2,5}$	12	$e_2(4,5)$	$\varnothing$	0
$\tau_{2,4}$	1	6	1	8	$\tau_{2,3}$	11	$e_2(2,6)$	$\varnothing$	0
$\tau_{2,5}$	1	11	1	10	$\tau_{2,3}$	13	$e_2(3,6)$	$\tau_{2,4}, \tau_{2,5}$	2
$\tau_{2,6}$	2	13	1	16	$\varnothing$	**16**	$e_2(5,6)$	$\tau_{2,3}$	3

Table 3.6, shows the results obtained by including first the effect of parallel and predecessor sub-tasks on each sub-task and their predecessors ($R^{pred}_{i,j}$) as we browse the graph. This table also presents the global response time $R^{global}_{i,j}$ that includes the effect of other parallel sub-tasks that are not predecessors.

We note that the response time of task τ_2 is reduced to $R^{global}_2 = R^{global}_{2,6} = 16$ instead of 18*. In fact, parallel sub-tasks on the critical path to the sink sub-task, like $\tau_{2,4}$ and $\tau_{2,5}$, are no longer included twice in the global response time of the sink sub-task $\tau_{2,5}$.*

3.2.2.4 Worst-case arrival patterns assumption used in previous analyses

Among different proposed response time equations, we try to reduce pessimism and enhance the estimate of the worst-case response time. Nevertheless, these approaches assume that critical instant and synchronous activation of all higher priority DAG tasks always occurs at each sub-task activation when browsing the graph. This context-independent assumption may not always be realistic and may cause an over-estimation of the worst-case response time. Through the following example, we show whether the assumption proposed by Palencia et al. [1] about worst-case arrivals pattern for dependent sub-tasks is always achievable and how it affects the computed upper-bound on response time.

Example 3.5. *Let us consider the task set composed of dependent sub-tasks on distributed systems described in Figure 3.5 and Table 3.7.*

If we apply the previous proposed response time Equations 3.18 and 3.19 that are based on the assumption about worst-case arrivals scenario proposed by Palencia et al. [1], we obtain the global response time estimations summarized in the following Table 3.8.

Table 3.7: Parameters of sub-tasks in Figure 3.5

Sub-task	$C_{i,j}$	T_i	Priority
$\tau_{1,1}$	30	80	high
$\tau_{2,1}$	3	100	low
$\tau_{2,2}$	5	100	low
$\tau_{2,3}$	3	100	low
$e_2(1,2)$	2	–	–
$e_2(2,3)$	1	–	–

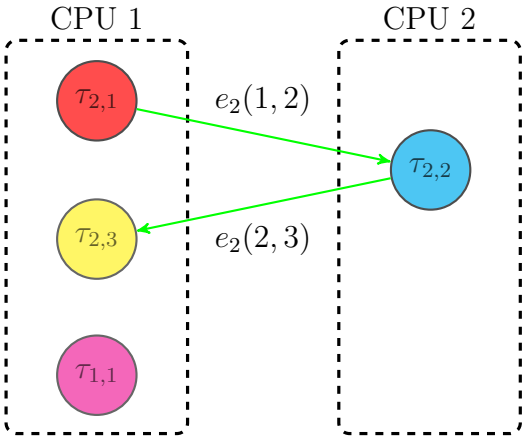
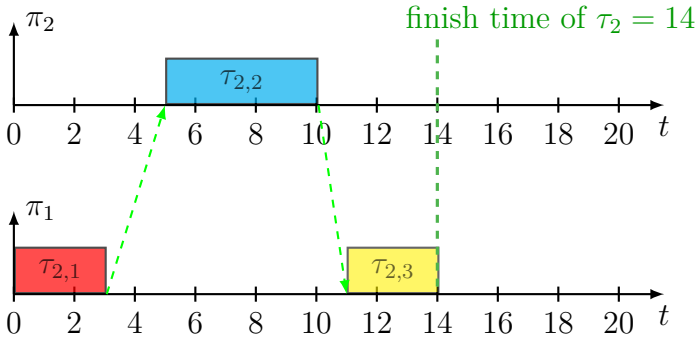


Figure 3.5: Example of a task set composed of dependent sub-tasks

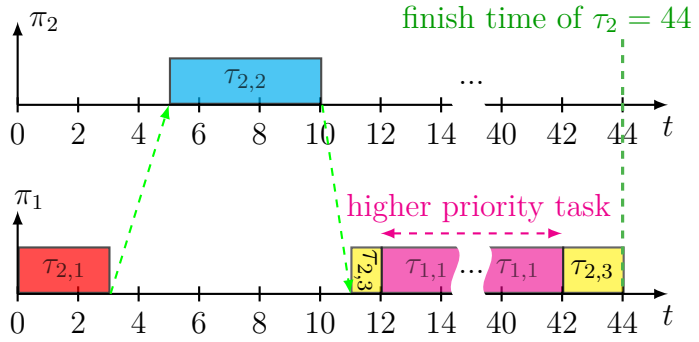
Table 3.8: Applying response time Equations 3.18 and 3.19 on the task set described in Figure 3.5

Sub-task	$C_{i,j}$	$J_{i,j}$	$I^{ext}(\tau_{i,j})$	$R^{global}_{i,j}$
$\tau_{1,1}$	30	0	0	30
$\tau_{2,1}$	3	0	30	33
$\tau_{2,2}$	5	35	0	40
$\tau_{2,3}$	3	41	30	74

We note that the estimated worst-case response time of task τ_2 *is* $R^{global}_{2,3} = 74$. *However, the scheduling in Figure 3.6 (on page 47) shows that the effect of the sub-task* $\tau_{1,1}$*, whenever it is released, will delay some or all sub-tasks of task* τ_1 *by* 30 *time units at the most. Besides,* τ_2 *will necessarily finish its execution before the next activation of* $\tau_{1,1}$ *and it cannot be preempted a second time. Hence, the actual response time of task* τ_2 *is* $R^{global}_{2,3} = 44$ *in the worst-case. We note that the calculated upper-bound on the response time is much bigger than the exact worst-case response time deduced from the scheduling in Figure 3.6. In summary, we conclude that the assumption made by Palencia et al. [1] about the worst-case scenario of higher priority sub-tasks activation is not always realistic, which may cause a lot pessimism in the worst-case response time estimation.*



(a) τ_2 executes in isolation (τ_1 is not activated)



(b) Worst case delay of DAG τ_2 if τ_1 is activated

Figure 3.6: Scheduling of the task set defined in Figure 3.5 and in Table 3.7

3.2.3 New characterization of worst-case arrival patterns

After observing the pessimism caused by the assumption about worst-case arrival patterns made by Palencia et al. [1], we suggest other ways to characterize the worst-case preempting workload of higher priority DAG tasks that delay the completion of the DAG task under study.

The first idea assumes that the sub-task under study is executed without any higher priority task (in isolation). To the obtained response time in isolation, we add the effect of higher priority tasks on the whole graph. The second idea is to include the effect of higher priority tasks not at the end after browsing the whole graph, nor at each sub-task activation but to include this effect on each connected sub-graph that executed on the same core and propagate it from each sub-task to its successors.

3.2.3.1 First method: Including preemption on the whole graph

This proposed method to define worst-case preemption caused by higher priority DAG tasks, is firstly based on the computation of the response time in isolation

(no higher priority tasks). In order to avoid including internal parallel sub-tasks several times, we proceed as we did previously in Section 3.2.2.3. We start by computing the preceding response time $R_{i,j}^{pred}$ of sub-tasks $\tau_{i,j}$ that considers the effect of predecessor sub-tasks but omits the effect of higher priority DAGs and parallel sub-tasks that are not predecessors.

Lemma 3.5. *The preceding response time $R_{i,j}^{pred}$ is computed as follows:*

$$R_{i,j}^{pred} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{pred} + e_i(k,j) + \mathbf{I_{i,j}^{pred}}(\tau_{\mathbf{i,k}}) \right\} + C_{i,j} \tag{3.21}$$

We denote by $I_{i,j}^{pred}(\tau_{i,k})$ the internal interference caused by predecessors of $\tau_{i,j}$ on $\tau_{i,k}$ and its predecessors. It is defined in Lemma 3.3 (on page 42).

Proof. Similarly to the proof of Lemma 3.4, we use mathematical induction to prove that the preceding response time (computed according to Equation 3.21) provides sufficient time for $\tau_{i,j}$ and all its predecessors to be executed without any preemption from parallel sub-tasks and higher priority DAGs.

■

Remark. *If each sub-task runs exactly for its WCET, then the preceding response time $R_{i,j}^{pred}$ could be seen as a lower bound of the response time (i.e. a Best Case Response Time) of the sub-task $\tau_{i,j}$. Indeed, $R_{i,j}^{pred}$ considers only the effect of predecessor sub-tasks that are required to finish their execution before $\tau_{i,j}$ could start its execution. Hence, whether the scenario of execution considered, $\tau_{i,j}$ cannot finish its execution before $R_{i,j}^{pred}$ time units from its release.*

Secondly, we compute the response time in isolation $R_{i,j}^{isol}$ that considers the effect of all sub-tasks from the same DAG on sub-task $\tau_{i,j}$ but omits preemptions caused by higher priority DAGs.

Lemma 3.6. *The response time in isolation is given by the following equation:*

$$R_{i,j}^{isol} = R_{i,j}^{pred} + \sum_{\tau_{\mathbf{i,k}} \in \mathbf{\Pi_{i,j}^{pred}}} \mathbf{C_{i,k}} \tag{3.22}$$

We denote by $\mathbf{\Pi_{i,j}^{pred}}$ the set of sub-tasks which are not predecessors of $\tau_{i,j}$ that could preempt $\tau_{i,j}$ or one of its predecessors. It is defined by Formula 3.20 (on page 44).

Proof. To obtain the response time in isolation $R_{i,j}^{isol}$ of sub-task $\tau_{i,j}$, we add to the preceding response time the delay caused by parallel sub-tasks on $\tau_{i,j}$ or on one of its predecessors. These parallel sub-tasks should not be predecessors to $\tau_{i,j}$ because their effect is already included in the preceding response time $R_{i,j}^{pred}$ through the interference term $I_{i,j}^{pred}(\tau_{i,k})$ in Equation 3.21.

Similarly to the proof of Lemma 3.3, we prove that the worst-case internal interference that sub-tasks of $\Pi_{i,j}^{pred}$ cause on $\tau_{i,j}$ and its predecessors, is equal to the sum of the execution times of all sub-tasks $\tau_{i,k} \in \Pi_{i,j}^{pred}$.

■

Finally, we compute the global response time that takes into consideration preemptions caused by sub-tasks in the same graph and by higher priority DAG tasks.

The global response time $R_{i,j}^{global}$ is calculated by summing the response time in isolation $R_{i,j}^{isol}$ and the external interference caused by higher priority DAG tasks on the whole graph (Equation 3.23).

Theorem 3.4. *The global response time is computed as follows:*

$$R_{i,j}^{global} = R_{i,j}^{isol} + \mathbf{I^{ext}}(\tau_{\mathbf{i,j}}) \tag{3.23}$$

The external interference $I^{ext}(\tau_{i,j})$, caused by higher priority preemption, is calculated by the following iterative equation on an interval of length $R_{i,j}^{isol}$:

$$I^{ext}(\tau_{i,j}) = \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{\mathbf{p,q}}) \in \pi(\mathbf{pred^\star}(\tau_{\mathbf{i,j}}))}} \left\lceil \frac{J_{p,q} + I^{ext}(\tau_{i,j}) + \mathbf{R_{i,j}^{isol}}}{T_p} \right\rceil C_{p,q} \tag{3.24}$$

Proof. The response time in isolation considers all possible preemptions except those caused by higher priority DAGs. Thus, we add to $R_{i,j}^{isol}$ the effect of other DAG tasks with higher priority (external interference) in order to obtain the global response time.

If a sub-task $\tau_{p,q}$, belonging to a higher priority DAG task, preempts a sub-task $\tau_{i,k} \in pred^\star(\tau_{i,j})$ (i.e. it preempts $\tau_{i,j}$ or one of its predecessors), then it delays not only $\tau_{i,k}$ but all sub-tasks $\tau_{i,a} \in pred^\star(\tau_{i,j})$. Hence, we consider that any sub-task belonging to a higher priority DAG will delay $\tau_{i,j}$ and all its predecessors if it is executed on the same core as a sub-task $\tau_{i,a} \in pred^\star(\tau_{i,j})$.

In order to estimate the worst-case external interference $I^{ext}(\tau_{i,j})$, we sum the delays caused by any higher priority sub-task $\tau_{p,q}$ that belong to another DAG task and that is executed on the same core as $\tau_{i,j}$ or one of its predecessors. Each delay is computed by an iterative equation, as in the literature [1], on an initial interval of length $R_{i,j}^{isol}$. Indeed, sub-task $\tau_{p,q}$ could preempt $\tau_{i,j}$ or one of its predecessors at each activation of $\tau_{p,q}$ in the interval $\left[0, R_{i,j}^{isol}\right[$.

■

Example 3.6. *In this example, we apply the proposed response time Equations 3.21, 3.22 and 3.23 on the task set defined in Figure 3.4 and Table 3.3. We show how taking into account higher priority preemption on the whole graph, instead of at each sub-task activation, allows us to avoid including frequent and non-realistic higher priority tasks activation and hence to reduce the computed worst-case response time.*

Table 3.9: Applying response time Equations 3.21, 3.22 and 3.23 on the task set described in Figure 3.4

Sub-task	$\mathbf{C_{i,j}}$	$\mathbf{R_{i,j}^{pred}}$	$\mathbf{\Pi_{i,j}^{pred}}$	$\mathbf{R_{i,j}^{isol}}$	$\mathbf{J_{i,j}}$	$\mathbf{I^{ext}}(\tau_{i,j})$	$\mathbf{R_{i,j}^{global}}$	**Edge**	$\mathbf{I_{i,j}^{pred}}(\tau_{i,k})$
$\tau_{1,1}$	3	3	$\varnothing$	3	0	0	3	$e_1(1,2)$	0
$\tau_{1,2}$	1	5	$\varnothing$	5	4	0	5	$e_2(1,2)$	0
$\tau_{2,1}$	2	2	$\varnothing$	2	0	3	5	$e_2(1,3)$	0
$\tau_{2,2}$	1	3	$\varnothing$	3	5	3	6	$e_2(1,4)$	0
$\tau_{2,3}$	3	6	$\tau_{2,4}, \tau_{2,5}$	8	6	4	12	$e_2(4,5)$	0
$\tau_{2,4}$	1	4	$\tau_{2,3}$	7	6	4	11	$e_2(2,6)$	0
$\tau_{2,5}$	1	5	$\tau_{2,3}$	8	11	4	**12**	$e_2(3,6)$	2
$\tau_{2,6}$	2	10	$\varnothing$	10	12	4	**14**	$e_2(5,6)$	3

From Table 3.9, we note that the response time of sub-task $\tau_{2,5}$ is equal to $R_{2,5}^{global} =$ 12 while the least pessimistic response time obtained by previous methods, based on the assumption of Palencia et al. [1], is equal to 13 *(obtained in Section 3.2.2.3). We note also that the global response time of DAG task τ_2 is reduced from* 16 *to* 14*. This improvement in the global response time is the result of including the effect of only the first activation of higher priority task τ_1 because, in this example, the next activation cannot occur when the current job studied task τ_2 is still running.*

We conclude that the proposed identification of the worst-case preempting workload implemented through Equations 3.21, 3.22 and 3.23 allows us to avoid including some non-realistic higher priority preemption. Hence, it may help to reduce the computed upper-bound of the response time and also to reduce the pessimism.

3.2.3.2 Second method: Including preemption on connected sub-graphs

After studying two approaches that characterize worst-case arrivals pattern of higher priority tasks, we present another way to identify this worst-case behavior. This new method is inspired by the two previous ones. However, it does not account for higher priority preemptions after browsing the whole graph neither at each sub-task activation like the assumption of Palencia et al. [1]. In fact, if several connected sub-tasks in a DAG task are executed on the same core then it is not likely that all higher priority task are synchronously activated at the release of

each connected sub-task especially when the response time in isolation of these sub-tasks is much smaller than the periods of preempting tasks.

On the other hand, taking into account higher priority preemptions on the whole graph over all core reduces the parallelism and increases the worst-case response time estimate. Therefore, we propose to compute the effect of higher priority tasks on different sub-graphs composed of connected sub-tasks mapped to the same core.

We define $G^{cnx}_{pred}(\tau_{i,j})$ as the set of sub-tasks belonging to the sub-graph composed of connected predecessors of sub-task $\tau_{i,j}$ that are executed on the same core as $\tau_{i,j}$.

$$\mathbf{G^{cnx}_{pred}}(\tau_{\mathbf{i,j}}) = \{\tau_{i,k} \in pred^{\star}(\tau_{i,j}) \mid \exists \text{ at least one directed path from } \tau_{i,k} \text{ to } \tau_{i,j} \text{ such that all the sub-tasks of this path are executed on core } \pi(\tau_{i,j})\} \tag{3.25}$$

First, we compute the preceding response time $R^{pred}_{i,j}$ of sub-tasks $\tau_{i,j}$ that takes into consideration the execution time $C_{i,j}$, the effect of all predecessor sub-tasks and the external interference caused by higher priority DAGs on predecessor sub-tasks that do not belong to connected sub-graph $G^{cnx}_{pred}(\tau_{i,j})$.

Lemma 3.7. *The preceding response time $R^{pred}_{i,j}$ is computed as follows:*

$$R^{pred}_{i,j} = C_{i,j} + \max_{\tau_{i,k} \in ipred(\tau_{i,j}} \left\{ R^{pred}_{i,k} + \mathbf{I_{i,j}}(\tau_{\mathbf{i,k}}) + e_i(k,j) + I^{pred}_{i,j}(\tau_{i,k}) \right\} \tag{3.26}$$

We denote by $I_{i,j}(\tau_{i,k})$ the external interference when the immediate predecessor $\tau_{i,k}$ is executed on a different core than $\tau_{i,j}$.

$$I_{i,j}(\tau_{i,k}) = \begin{cases} \mathbf{I^{ext}}(\tau_{\mathbf{i,k}}) & \text{if } \pi(\tau_{i,k}) \neq \pi(\tau_{i,j}) \\ 0 & \text{otherwise} \end{cases}$$

The external interference $I^{ext}(\tau_{i,j})$ accounts for the effect of the higher priority DAGs on sub-task $\tau_{i,j}$ and its predecessors belonging to the connected sub-graph $G^{cnx}_{pred}(\tau_{i,j})$ and mapped to the same core as $\tau_{i,j}$.

Proof. Similarly to the proof of Lemma 3.4, we use mathematical induction to prove that the preceding response time (computed according to Equation 3.26) provides sufficient time for the execution of $\tau_{i,j}$, all its predecessors and higher priority DAGs that preempt predecessors not in the connected sub-graph $G^{cnx}_{pred}(\tau_{i,j})$.

■

Lemma 3.8. *The worst-case external interference, exerted on $\tau_{i,j}$ and its predecessors belonging to $G^{cnx}_{pred}(\tau_{i,j})$, is given as follows:*

$$I^{ext}(\tau_{i,j}) = \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{J_{p,q} + I^{ext}(\tau_{i,j}) + \mathbf{I^{cnx}}(\tau_{\mathbf{i,j}})}{T_p} \right\rceil C_{p,q} \tag{3.27}$$

We consider that the initial interval $I^{cnx}(\tau_{i,j}) =$ used for computing this external interference is computed as follows:

$$I^{cnx}(\tau_{i,j}) = \sum_{\tau_{\mathbf{i,k}} \in \mathbf{G^{cnx}_{pred}}(\tau_{\mathbf{i,j}})} \mathbf{C_{i,k}} + \sum_{\tau_{\mathbf{i,k}} \in \mathbf{\Pi^{cnx}_{i,j}}} \mathbf{C_{i,k}} \tag{3.28}$$

The set $\Pi^{cnx}_{i,j}$ is composed of sub-tasks that are parallel to one sub-task in $\mathbf{G^{cnx}_{pred}}(\tau_{\mathbf{i,j}})$ and that run on same core and have higher or equal priority. We recall that all sub-tasks in $\mathbf{G^{cnx}_{pred}}(\tau_{\mathbf{i,j}})$ are mapped to the same core as $\tau_{i,j}$.

$$\mathbf{\Pi^{cnx}_{i,j}} = \{\tau_{i,k} \in V_i \setminus G^{cnx}_{pred}(\tau_{i,j}) \mid \exists\ \tau_{i,l} \in G^{cnx}_{pred}(\tau_{i,j})\ \textit{such that}\ \tau_{i,k} \in P_{i,l}\} \tag{3.29}$$

Proof. Similarly to the proof of Theorem 3.4, to estimate the worst-case external interference $I^{ext}(\tau_{i,j})$, we sum the delays caused by any higher priority sub-task $\tau_{p,q}$ that belongs to another DAG task and that are mapped to the same core as $\tau_{i,j}$.

Each delay is computed by an iterative equation, as in the literature [1], on an initial time window of length $I^{cnx}(\tau_{i,j})$. Indeed, sub-task $\tau_{p,q}$ could preempt a sub-task $\tau_{i,k} \in G^{cnx}_{pred}(\tau_{i,j})$ at each activation of $\tau_{p,q}$ in the interval $[0, I^{cnx}(\tau_{i,j})[$ of length the sum of execution times of sub-tasks in $G^{cnx}_{pred}(\tau_{i,j})$ and their parallel sub-tasks in $\Pi^{cnx}_{i,j}$. Hence, we express $I^{cnx}(\tau_{i,j})$ by Equation 3.28.

■

In order to obtain the global response time of sub-task $\tau_{i,j}$, we add to the preceding response time $R^{pred}_{i,j}$ the effect of parallel sub-tasks that are not predecessors of $\tau_{i,j}$ as well as the effect of external interference $I^{ext}(\tau_{i,j})$ exerted on any sub-task $\tau_{i,k} \in G^{cnx}_{pred}(\tau_{i,j})$.

Theorem 3.5. *The global response time is computed by the following equation:*

$$R^{global}_{i,j} = R^{pred}_{i,j} + \sum_{\tau_{\mathbf{i,k}} \in \mathbf{\Pi^{pred}_{i,j}}} \mathbf{C_{i,k}} + \mathbf{I^{ext}}(\tau_{\mathbf{i,j}}) \tag{3.30}$$

We denote by $\mathbf{\Pi^{pred}_{i,j}}$ the set of sub-tasks not predecessors of $\tau_{i,j}$ that could preempt $\tau_{i,j}$ or one of its predecessors. It is defined by Formula 3.20 (on page 44).

Proof. Similarly to the proof of Lemma 3.3, we prove that the worst-case internal interference cause by parallel and not predecessor sub-tasks (composing $\Pi^{pred}_{i,j}$) on $\tau_{i,j}$ and its predecessors, is equal to the sum of the execution times of all these sub-tasks in $\Pi^{pred}_{i,j}$.

If a sub-task $\tau_{p,q}$ belongs to a higher priority DAG task and runs on the same core as $\tau_{i,j}$ or as one of its predecessors, it delays the finish of $\tau_{i,j}$. Indeed, $\tau_{p,q}$ preempts a sub-task $\tau_{i,k} \in pred^{\star}(\tau_{i,j})$. There are two possible cases; If $\tau_{i,k} \notin G^{cnx}_{pred}(\tau_{i,j})$, then

the delay caused by $\tau_{p,q}$ is already computed in the preceding response time $R_{i,j}^{pred}$. Otherwise, if $\tau_{i,k} \in G_{pred}^{cnx}(\tau_{i,j})$, then the delay caused by $\tau_{p,q}$ is computed in $\mathbf{I^{ext}(\tau_{i,j})}$. Thus, we should add $I^{ext}(\tau_{i,j})$ to the preceding response time $R_{i,j}^{pred}$ in order to get the global response time.

■

Example 3.7. *On the task set example defined by Figure 3.4 and Table 3.3, we apply the previous response time Equations 3.26 and 3.30 that include higher priority preemptions on connected sub-graphs instead of including them at each sub-task activation. We illustrate how this new approach avoids the accounting for frequent and non-realistic higher priority tasks activation and how it reduces the pessimism.*

Table 3.10: Parameters of sub-tasks in Figure 3.1

Sub-task	$\mathbf{C_{i,j}}$	$\mathbf{R_{i,j}^{pred}}$	$\mathbf{G_{pred}^{cnx}(\tau_{i,j})}$	$\mathbf{\Pi_{i,j}^{cnx}}$	$\mathbf{I^{cnx}(\tau_{i,j})}$	$\mathbf{J_{i,j}}$	$\mathbf{I^{ext}(\tau_{i,j})}$	$\mathbf{R_{i,j}^{global}}$
$\tau_{1,1}$	3	3	$\tau_{1,1}$	$\varnothing$	3	0	0	3
$\tau_{1,2}$	1	5	$\tau_{1,2}$	$\varnothing$	1	4	0	5
$\tau_{2,1}$	2	2	$\tau_{2,1}$	$\varnothing$	2	0	3	5
$\tau_{2,2}$	1	3	$\tau_{2,1}, \tau_{2,2}$	$\varnothing$	5	3	3	6
$\tau_{2,3}$	3	9	$\tau_{2,3}$	$\tau_{2,4}, \tau_{2,5}$	5	6	1	12
$\tau_{2,4}$	1	7	$\tau_{2,4}$	$\tau_{2,3}$	4	6	1	11
$\tau_{2,5}$	1	8	$\tau_{2,4}, \tau_{2,5}$	$\tau_{2,3}$	6	11	1	**12**
$\tau_{2,6}$	2	13	$\tau_{2,3}, \tau_{2,4}, \tau_{2,5}, \tau_{2,6}$	$\varnothing$	8	12	1	**14**

In Table 3.10, we present the results of applying the proposed Equations 3.26 and 3.30. We note that the response time of task τ_2 is reduced to $R_{2,6}^{global} = 14$ compared to the analyses based on the assumption of Palencia et al. [1] about worst-case preemption scenario. Hence, this approach allows us to reduce the computed upper-bound of response time by not including some non-achievable arrivals of higher priority tasks. Although, we obtain the same result ($R_{2,6}^{global} = 14$) as the previous proposed characterization of worst-case arrivals pattern, we cannot deduce that the two approaches are equivalent. In fact, we will see, in Chapter 6, that they may yield different results for other task sets with different numbers of tasks, cores, mapping, etc.

We conclude that this approach of identifying the worst-case preempting workload implemented through Equations 3.26 and 3.30 avoids taking into account non-realistic higher priority preemption. Therefore, it reduces the computed upper-bound of the response time as well as the pessimism. However, we should keep in mind that the two previously proposed approaches for characterizing the worst-case preemption scenario are not equivalent.

3.3 Conclusion

In this chapter, we presented a schedulability analysis based on RTA for a hard real-time system composed of DAG tasks. We addressed the case of deterministic timing parameters and we considered a partitioned, fixed-priority and preemptive scheduling policy on multi-core processors. We extended existing response time equations for distributed systems [1] that operates on chains of sub-tasks without any possible parallelism inside a chain. We proposed several response time analyses to deal with a DAG task model with parallel sub-tasks inside the same task. These analyses are based on different approaches to upper bound the internal and external interferences. In Table 3.11 (on page 55), we present a summary of all RTA approaches proposed in this chapter.

First, we included the effect of parallel sub-tasks in the local response time of the sub-task studied. This approach may be pessimistic because in some cases it considers the same sub-task several times if it is parallel to several other sub-tasks.

Second, we proposed to take into account the parallel sub-tasks in the global response time instead of in the local response time. This method allows us to avoid including some sub-tasks several times, but it takes into account the execution time of some sub-tasks twice in specific cases.

Third, we provided an approach in-between that accounts a part of the parallel workload belonging to the predecessor sub-tasks in the local response time. While, the remaining parallel sub-tasks are included in the global response time. This approach avoids considering some sub-tasks more than once when computing the internal interference. Similarly to the previous methods, this method is also based on the characterization of the worst-case arrival patterns of higher priority tasks described in [1] when computing the external interference. We showed that this characterization may be pessimistic in some cases.

Therefore, we provided two new approaches to characterize the worst-case arrival patterns. The first one starts by computing the response time in isolation that omits the effect of any higher priority DAG task. Then, it takes into account higher priority preemptions, from different cores, on the whole graph. The second method considers the effect of higher priority DAG tasks on connected sub-graphs that are executed on the same core. These two approaches, allow to reduce the response time compared to the three previous methods in most of cases.

For all these methods, we provided theoretical proofs to guarantee the safety of their response time equations (i.e. never under-estimate the actual response time). Thus, the corresponding schedulability tests are sufficient to ensure the

Table 3.11: Summary of different response time equations proposed in this chapter.

Methods	Response time equations
Holistic extension 1 (Sec. 3.2.2.1)	$I_i^{int}(\tau_{i,j}) = \sum_{\tau_{i,k} \in P_{i,j}} C_{i,k}$ $I^{ext}(\tau_{i,j}) = \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{J_{p,q} + I^{ext}(\tau_{i,j}) + C_{i,j} + I_i^{int}(\tau_{i,j})}{T_p} \right\rceil C_{p,q}$ $J_{i,j} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{global} + e_i(k,j) \right\}$ $R_{i,j}^{global} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{global} + e_i(k,j) \right\} + I_i^{int}(\tau_{i,j}) + I^{ext}(\tau_{i,j})$
Holistic extension 2 (Sec. 3.2.2.2)	$R_{i,j}^{seq} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{seq} + e_i(k,j) \right\} + C_{i,j} + I^{ext}(\tau_{i,j})$ $R_{i,j}^{global} = R_{i,j}^{seq} + \sum_{\tau_{i,k} \in \Pi_{i,j}} C_{i,k}$
Holistic extension 3 (Sec. 3.2.2.3)	$R_{i,j}^{pred} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{pred} + e_i(k,j) + I_{i,j}^{pred}(\tau_{i,k}) \right\} + C_{i,j} + I^{ext}(\tau_{i,j})$ $I_{i,j}^{pred}(\tau_{i,k}) = \sum_{\tau_{i,l} \in \Psi_{i,j}(\tau_{i,k})} C_{i,l}$ $R_{i,j}^{global} = R_{i,j}^{pred} + \sum_{\tau_{i,k} \in \Pi_{i,j}^{pred}} C_{i,k}$
Our method 1 (Sec. 3.2.3.1)	$R_{i,j}^{pred} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ R_{i,k}^{pred} + e_i(k,j) + I_{i,j}^{pred}(\tau_{i,k}) \right\} + C_{i,j}$ $R_{i,j}^{isol} = R_{i,j}^{pred} + \sum_{\tau_{i,k} \in \Pi_{i,j}^{pred}} C_{i,k}$ $R_{i,j}^{global} = R_{i,j}^{isol} + I^{ext}(\tau_{i,j})$ $I^{ext}(\tau_{i,j}) = \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) \in \pi(pred^{\star}(\tau_{i,j}))}} \left\lceil \frac{J_{p,q} + I^{ext}(\tau_{i,j}) + R_{i,j}^{isol}}{T_p} \right\rceil C_{p,q}$
Our method 2 (Sec. 3.2.3.1)	$R_{i,j}^{pred} = C_{i,j} + \max_{\tau_{i,k} \in ipred(\tau_{i,j}} \left\{ R_{i,k}^{pred} + I_{i,j}(\tau_{i,k}) + e_i(k,j) + I_{i,j}^{pred}(\tau_{i,k}) \right\}$ $I_{i,j}(\tau_{i,k}) = I^{ext}(\tau_{i,j})$ if $\pi(\tau_{i,j}) \neq \pi(\tau_{i,k})$, otherwise 0 $I^{ext}(\tau_{i,j}) = \sum_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{J_{p,q} + I^{ext}(\tau_{i,j}) + I^{cnx}(\tau_{i,j})}{T_p} \right\rceil C_{p,q}$ $I^{cnx}(\tau_{i,j}) = \sum_{\tau_{i,k} \in G_{pred}^{cnx}(\tau_{i,j})} C_{i,k} + \sum_{\tau_{i,k} \in \Pi_{i,j}^{cnx}} C_{i,k}$ $R_{i,j}^{global} = R_{i,j}^{pred} + \sum_{\tau_{i,k} \in \Pi_{i,j}^{pred}} C_{i,k} + I^{ext}(\tau_{i,j})$

feasibility of the hard real-time system studied. Moreover, in Chapter 6, we study and compare the performance of the 5 methods proposed in this chapter. We will see that most of them are not comparable and that their run-times are reasonable compared to other existing analysis due to the polynomial complexity of iterative equations. Hence, we propose a schedulability test that combines all these methods together in order to benefit from the performance of each of them. Indeed, we could apply all these approaches to estimate the response time of a given real-time system. Then, we use the minimum response time estimation for the schedulability test. Since all these methods provide a safe estimation of the response time, then this minimum is also safe and it never under-estimates the actual response time. Thus, this combined schedulability test is also sufficient to guarantee the feasibility of hard real-time systems

4

Probabilistic DAG Tasks Schedulability on a Multi-core Processor

Contents

The execution time of a program (sub-task) varies depending on many factors such as the input values, the path taken through the code and the state of hardware components (such as cache memory and communication bus). This variability may induce important additional time in the WCET, because of conditions that rarely occur. For a more accurate analysis, we model execution times and communication delays by probability distributions where we associate a probability to each possible value of execution or communication time.

In this chapter, we study the schedulability of a DAG task model with probabilistic execution times and communication delays. We consider the two proposed

RTA (Sections 3.2.3.1 and 3.2.3.2) that characterize the worst-case arrival patterns differently from the existing work of Palencia et al. [1]. Indeed, we adapt these response time equations to deal with probabilistic parameters. Firstly, we simplify the analysis and keep it scalable by assuming that all random variables used are independent. However, this assumption does not always hold. Secondly, therefore, we use a Bayesian network to model the dependencies between the different random variables employed in the response time equations.

In addition, we study the schedulability of a probabilistic DAG task model among C-space [83, 84] that represents different possible values of each execution time distribution. We use a machine learning classification technique called SVM (Support Vector Machine) [85, 86] to determine the combinations of values that lead to a schedulable system. Then, we weight each combination with its probability of occurrence and we sum these probabilities to deduce the probability of the system being schedulable.

4.1 Probabilistic Task Model and Definitions

In this chapter, we consider a task model similar to the DAG task model used in the previous Chapter 3 and we add probabilistic timing parameters like execution times and communication delays. We denote by pWCET (respectively pWCCT), the probabilistic Worst-Case Execution Time (respectively the probabilistic Worst-Case Communication Time) as defined below.

Definition 4.1 (From Davis and Cucu [87])**.** *The pWCET distribution of a program is the least upper bound, in the sense of the greater than or equal to operator* $\succeq$ *(see Definition 4.3), on the execution time distribution of the program for every valid scenario of operation, where a scenario of operation is defined as an infinitely repeating sequence of input states and initial hardware states that characterize a feasible way in which recurrent execution of the program may occur.*

Definition 4.2. *Similarly, we define pWCCT distribution as the least upper bound of communication delay distribution between two programs (sub-tasks) for every valid scenario of operation.*

Definition 4.3 (From Diaz et al. [88])**.** *The probability distribution of a random variable* $\mathcal{X}$ *is said to be greater than or equal to (i.e. upper bounds) another random variable* $\mathcal{Y}$ *and we denote* $\mathcal{X} \succeq \mathcal{Y}$*, if* $\forall\ x \in \mathbb{R},\ F_{\mathcal{X}}(x) \leq F_{\mathcal{Y}}(x)$*, (i.e.* $1 - F_{\mathcal{X}}(x) \geq 1 - F_{\mathcal{Y}}(x)$*).*

Remark. *In the remainder, we use calligraphic characters, such as $\mathcal{X}$, to denote random variables.*

Graphically, this means that the Cumulative Distribution Function (CDF) of $\mathcal{X}$ is always below that of $\mathcal{Y}$, or alternatively, the 1-CDF (exceedance function) of $\mathcal{X}$ is always above that of $\mathcal{Y}$ (see curves in Figure 4.1). We note that the greater than or equal to relation $\succeq$ between two random variables does not provide a total order. For instance, in Figure 4.1, the 1-CDF curves of random variables $\mathcal{X}$ and $\mathcal{Z}$ cross. Then, we say that $\mathcal{X}$ and $\mathcal{Z}$ are not comparable, i.e. $\mathcal{X} \not\succeq \mathcal{Z}$ and $\mathcal{Z} \not\succeq \mathcal{X}$.

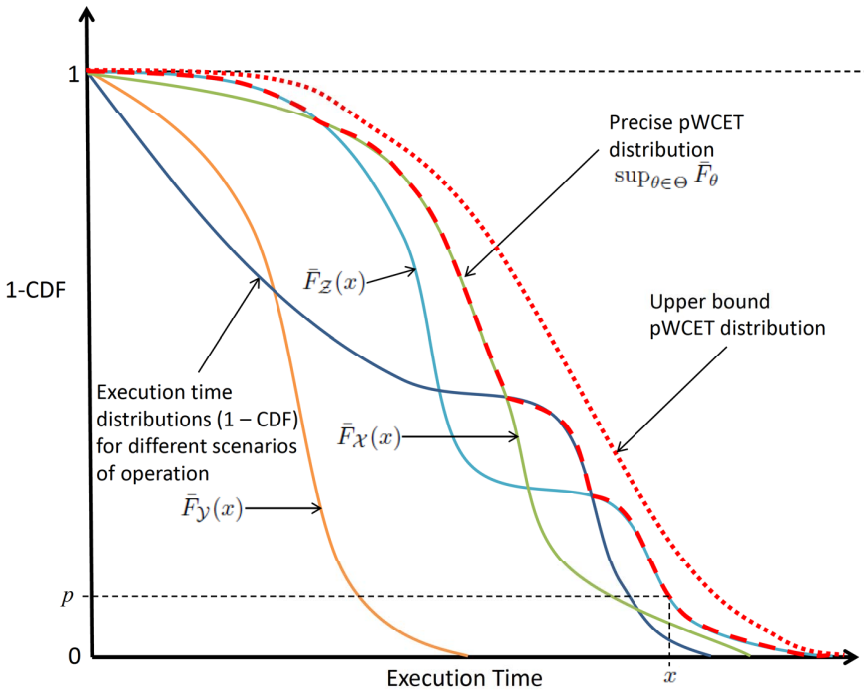


Figure 4.1: Example for the pWCET distribution of a program, and also execution time distributions for specific scenarios of operation as presented by Davis and Cucu [87]

In Figure 4.1, solid lines represent execution time distributions of the program (sub-task) for different scenarios of operation. The red dashed line is the least upper bound (higher envelope) of all these distributions. It represents the actual pWCET distribution of the program over every valid scenario of operation, while the red dotted line is a pessimistic estimate (upper bound) of the actual pWCET. Exploring all possible scenarios of operation to determine the actual pWCET is often impractical. Thus, we use an estimate of pWCET to characterize a sub-task. This estimate should be equal to or upper bound (overestimate) the actual pWCET to guarantee that it is a safe estimate but this overestimation introduces some pessimism. Hence, a tighter upper bound of pWCET allows us to reduce pessimism while being safe.

Notation for the probabilistic parameters

In the remainder, to represent a probabilistic parameter (pWCET, pWCCT), we use a **discrete** random variable that has a **finite** number of possible values. It is defined by its probability mass function (discrete distribution) as follows:

Definition 4.4. *The probability mass function $f_{\mathcal{C}}$ of the discrete random variable $\mathcal{C}$, that has $K_{\mathcal{C}}$ possible values denoted $\left\{C^h,\ 1 \leq h \leq K_{\mathcal{C}}\right\}$, is defined by:*

$$f_{\mathcal{C}}\left(C^h\right) = P\left(\mathcal{C} = C^h\right),\ \forall h \in \{1, \ldots, K_{\mathcal{C}}\}$$

We associate possible values of $\mathcal{C}$ with their probabilities according to the following notation:

$$\mathcal{C} = \begin{pmatrix} C^1 & C^2 & \cdots & C^{K_{\mathcal{C}}} \\ f_{\mathcal{C}}(C^1) & f_{\mathcal{C}}(C^2) & \cdots & f_{\mathcal{C}}(C^{K_{\mathcal{C}}}) \end{pmatrix}$$

where $C^h < C^{h+1},\ \forall h \in \{1, 2, \ldots, K_{\mathcal{C}} - 1\}$. Thus, C^1 is the minimum value that the random variable $\mathcal{C}$ could take and $C^{K_{\mathcal{C}}}$ is its maximum value. We note that $\sum_{h=1}^{K_{\mathcal{C}}} f_{\mathcal{C}}\left(C^h\right) = 1$ according to the definition of probability mass function.

We denote the pWCET of sub-task $\tau_{i,j}$ by $\mathcal{C}_{i,j}$ and the pWCCT between sub-tasks $\tau_{i,j}$ and $\tau_{i,k}$ by $\mathcal{E}_i(j,k)$.

4.2 Extension of Response Time Equations

In this section, we extend the deterministic RTA proposed in the previous Chapter 3 to deal with probabilistic execution and communication times. The probabilistic RTA equations should estimate the distribution of probabilistic Worst-Case Response time (pWCRT) instead of a single WCRT value for each sub-task and DAG task. The pWCRT distribution of a DAG task is used to derive the Deadline Miss Probability (DMP), as defined below, of that DAG task instead of a binary decision about the respect or not of the deadline, as considered in the deterministic analysis.

Definition 4.5. *The Deadline Miss Probability DMP_i of task τ_i is the probability that any job of task τ_i misses its deadline. It is equal to the probability that the pWCRT of task τ_i exceeds its deadline D_i:*

$$DMP_i = P(\mathcal{R}_i^{global} > D_i)$$

We denote by $\mathcal{R}_i^{global}$ the pWCRT distribution of task τ_i as defined by Equation 3.13 (on page 38).

In this section, we assume that the random variables used like pWCET and pWCCT are **independent** in order to simplify the analysis as a first step towards a more general one. Hence, in the next Section 4.3, we study the case of dependent random variables and we establish a more precise analysis based on a Bayesian network to model existing dependencies.

Definition 4.6 (From Davis and Cucu [87])**.** *Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are independent if they describe two events such that the knowledge of whether one event did or did not occur does not change the probability that the other event occurs.*

Stated otherwise, the joint probability is equal to the product of their probabilities:

$$P\left(\{\mathcal{X}=x\} \cap \{\mathcal{Y}=y\}\right) = P\left(\mathcal{X}=x\right) \cdot P\left(\mathcal{Y}=y\right)$$

In our context, a random variable characterizes the event that the execution times of a program or the communication times between two programs takes a certain value.

4.2.1 Probabilistic Operators

A probabilistic operator manipulates probability distributions of random variables and it produces the probability distribution of the random variable defined by this operator. In order to adapt the response time equations developed in Sections 3.2.3.1 and 3.2.3.2 to our probabilistic analysis, we present the required probabilistic operators.

4.2.1.1 Probabilistic sum (convolution) operator

The convolution operator is used to compute the sum of two independent random variables as defined below:

Definition 4.7. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two discrete and **independent** random variables and $\mathcal{Z}$ the random variable defined by their sum $\mathcal{Z} = \mathcal{X} + \mathcal{Y}$.*

*The probability mass function (distribution) $f_{\mathcal{Z}}$ of $\mathcal{Z}$ is defined by the **convolution** of probability distribution of $\mathcal{X}$ and $\mathcal{Y}$ i.e. $f_{\mathcal{Z}} = f_{\mathcal{X}} \otimes f_{\mathcal{Y}}$. More explicitly, $f_{\mathcal{Z}}$ is defined as follows:*

$$f_{\mathcal{Z}}(t) = P\left(\mathcal{Z}=t\right) = \sum_{k=-\infty}^{k=+\infty} P\left(\mathcal{X}=k\right) \cdot P\left(\mathcal{Y}=t-k\right) \tag{4.1}$$

Example 4.1. *For instance, the convolution of two probability distributions is as follows:*

$$\begin{pmatrix} 3 & 7 \\ 0.3 & 0.7 \end{pmatrix} \otimes \begin{pmatrix} 0 & 4 \\ 0.1 & 0.9 \end{pmatrix} = \begin{pmatrix} 3 & 7 & 11 \\ 0.3 \times 0.1 & 0.3 \times 0.9 + 0.7 \times 0.1 & 0.7 \times 0.9 \end{pmatrix}$$
$$= \begin{pmatrix} 3 & 7 & 11 \\ 0.03 & 0.34 & 0.63 \end{pmatrix}$$

Remark. *For the sake of the simplicity, in the remainder of this thesis, we denote the probability distribution $f_{\mathcal{X}}$ by the name of its random variable i.e. $\mathcal{X}$. Thus, we refer to the convolution of two probability distributions by $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y}$ (instead of $f_{\mathcal{Z}} = f_{\mathcal{X}} \otimes f_{\mathcal{Y}}$).*

A complementary operator to the convolution is the operator $\ominus$, defined by $\mathcal{X} \ominus \mathcal{Y} = \mathcal{X} \otimes (-\mathcal{Y})$.

4.2.1.2 Probabilistic maximum operator

The maximum operator operates on probability distributions and computes a probability distribution of a random variable that is greater than or equal to all the provided distributions. The greater than or equal to relation could be defined on values taken by independent random variables or it could also be defined on the CDF function (see Definition 4.3). Depending on the greater than or equal to relation used, we can define different max operators.

Probabilistic maximum based on independent random variables comparison

The random variable defined as the maximum of two independent random variables has a probability distribution defined as below:

Definition 4.8. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two discrete and **independent** random variables and $\mathcal{Z} = \max(\mathcal{X}, \mathcal{Y})$*

The cumulative distribution function $F_{\mathcal{Z}}$ of $\mathcal{Z}$ is computed as follows:

$$\begin{aligned} F_{\mathcal{Z}}(t) = P(\mathcal{Z} \leqslant t) &= P(\max(\mathcal{X}, \mathcal{Y}) \leqslant t) \\ &= P(\mathcal{X} \leqslant t, \mathcal{Y} \leqslant t) \\ &= P(\mathcal{X} \leqslant t) \cdot P(\mathcal{Y} \leqslant t) \quad (\textit{independence}) \\ &= \sum_{i=-\infty}^{t} P(\mathcal{X} = i) \cdot \sum_{j=-\infty}^{t} P(\mathcal{Y} = j) \end{aligned}$$

The probability distribution $f_{\mathcal{Z}}$ of $\mathcal{Z}$ is given by:

$$f_{\mathcal{Z}}(t) = P(\mathcal{Z} = t) = \sum_{\max(i,j)=t} P(\mathcal{X} = i) \cdot P(\mathcal{Y} = j) \tag{4.2}$$

We define the probabilistic maximum operator of independent probability distribution $f_{\mathcal{X}}$ and $f_{\mathcal{Y}}$ by:

$$\mathcal{M}\text{ax}_{\text{Indep}}\{f_{\mathcal{X}}, f_{\mathcal{Y}}\} = f_{\mathcal{Z}}$$

We note that $\mathcal{Z} \succeq \mathcal{X}$ because $F_{\mathcal{Z}}(t) = F_{\mathcal{X}}(t) \cdot F_{\mathcal{Y}}(t) \leq F_{\mathcal{X}}(t),\ \forall\ t \in \mathbb{R}$ (since $0 \leq F_{\mathcal{X}}(t) \leq 1$ and $0 \leq F_{\mathcal{Y}}(t) \leq 1,\ \forall\ t \in \mathbb{R}$). Similarly, we deduce that $\mathcal{Z} \succeq \mathcal{Y}$.

Remark. *For the sake of simplicity of notation, in the remainder, we denote the maximum operator of independent probability distribution by $\mathcal{M}\text{ax}_{\text{Indep}}\{\mathcal{X}, \mathcal{Y}\} = \mathcal{Z}$ (instead of $\mathcal{M}\text{ax}_{\text{Indep}}(f_{\mathcal{X}}, f_{\mathcal{Y}}) = f_{\mathcal{Z}}$), where $\mathcal{Z}$ is the probability distribution computed above (in Equation 4.2).*

Example 4.2. *For instance, the maximum between two independent probability distributions is as follows:*

$$\begin{aligned}\mathcal{M}\text{ax}_{\text{Indep}}\left\{\begin{pmatrix}3 & 7\\ 0.3 & 0.7\end{pmatrix}, \begin{pmatrix}0 & 4\\ 0.1 & 0.9\end{pmatrix}\right\} &= \begin{pmatrix}3 & 4 & 7\\ 0.3 \times 0.1 & 0.3 \times 0.9 & 0.7 \times 0.1 + 0.7 \times 0.9\end{pmatrix}\\ &= \begin{pmatrix}3 & 4 & 7\\ 0.03 & 0.27 & 0.7\end{pmatrix}\end{aligned}$$

Probabilistic maximum based on CDF function comparison

According to the comparison relation of CDF functions (defined in Definition 4.3), the maximum of probability distributions, also called the supremum, is defined in the work of Diaz et al. [2].

Definition 4.9 (From Diaz et al. [2])**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be two discrete random variables, we denote their supremum (maximum) by the random variable $\mathcal{Z} = \mathcal{M}\text{ax}_{\text{Diaz}}\{\mathcal{X}, \mathcal{Y}\}$. $\mathcal{Z}$ is defined with its CDF function given below:*

$$F_{\mathcal{Z}}(t) = \min\left(F_{\mathcal{X}}(t), F_{\mathcal{Y}}(t)\right),\ \forall\ t \in \mathbb{R}$$

From the definition of $\mathcal{Z}$, we note that $\mathcal{Z} \succeq \mathcal{X}$ because $F_{\mathcal{Z}}(t) = \min\left(F_{\mathcal{X}}(t), F_{\mathcal{Y}}(t)\right) \leq F_{\mathcal{X}}(t),\ \forall\ t \in \mathbb{R}$. Similarly, we deduce that $\mathcal{Z} \succeq \mathcal{Y}$.

Graphically, the maximum, defined by Diaz et al. [2], of two probability distributions is the higher envelope of their 1-CDF functions (i.e. the lower envelope of their CDF functions) as illustrated in Figure 4.1 on page 59.

Example 4.3. *For instance, the maximum of two probability distributions, according to the definition of Diaz et al. [2], is computed as follows:*

$$\mathcal{M}\text{ax}_{\text{Diaz}}\{\mathcal{X},\mathcal{Y}\} = \mathcal{M}\text{ax}_{\text{Diaz}}\left\{\begin{pmatrix}3 & 7\\ 0.3 & 0.7\end{pmatrix},\begin{pmatrix}0 & 4\\ 0.1 & 0.9\end{pmatrix}\right\} = \begin{pmatrix}3 & 4 & 7\\ 0.1 & 0.2 & 0.7\end{pmatrix}$$

In fact, the corresponding CDF functions are $\begin{pmatrix}0 & 3 & 4 & 7\\ 0 & 0.3 & 0.3 & 1\end{pmatrix}$ *and* $\begin{pmatrix}0 & 3 & 4 & 7\\ 0.1 & 0.1 & 1 & 1\end{pmatrix}$. *Thus, the CDF function of their supremum (maximum) is:*

$$\begin{pmatrix}0 & 3 & 4 & 7\\ \min(0,0.1) & \min(0.3,0.1) & \min(0.3,1) & \min(1,1)\end{pmatrix} = \begin{pmatrix}0 & 3 & 4 & 7\\ 0 & 0.1 & 0.3 & 1\end{pmatrix}$$

Graphically, we note, from the Figure 4.2 (on page 66), that the $\mathcal{M}\text{ax}_{\text{Diaz}}\{\mathcal{X},\mathcal{Y}\}$ *(the red dashed line) is the lower envelope of the CDF functions (i.e. higher envelope 1-CDF functions) of random variables* $\mathcal{X}$ *and* $\mathcal{Y}$.

Lemma 4.1. *Let* $\mathcal{X}$ *and* $\mathcal{Y}$ *be two discrete and independent random variables. Then:*

$$\mathcal{M}\text{ax}_{\text{Indep}}\{\mathcal{X},\mathcal{Y}\} \succeq \mathcal{M}\text{ax}_{\text{Diaz}}\{\mathcal{X},\mathcal{Y}\}$$

Proof. The CDF function of $\mathcal{M}\text{ax}_{\text{Indep}}\{\mathcal{X},\mathcal{Y}\}$ is equal to $F_{\mathcal{X}}(t)\cdot F_{\mathcal{Y}}(t),\ \forall\ t\in\mathbb{R}$. While, the CDF function of $\mathcal{M}\text{ax}_{\text{Diaz}}\{\mathcal{X},\mathcal{Y}\}$ is equal to $\min\left(F_{\mathcal{X}}(t),F_{\mathcal{Y}}(t)\right),\ \forall\ t\in\mathbb{R}$.

We know that $0\leq F_{\mathcal{X}}(t)\leq 1$ and $0\leq F_{\mathcal{Y}}(t)\leq 1,\ \forall\ t\in\mathbb{R}$. Thus, $F_{\mathcal{X}}(t)\cdot F_{\mathcal{Y}}(t)\leq F_{\mathcal{X}}(t),\ \forall\ t\in\mathbb{R}$ (since $F_{\mathcal{Y}}(t)\leq 1$ and $0\leq F_{\mathcal{X}}(t),\ \forall\ t\in\mathbb{R}$). Similarly, we can prove that $F_{\mathcal{X}}(t)\cdot F_{\mathcal{Y}}(t)\leq F_{\mathcal{Y}}(t),\ \forall\ t\in\mathbb{R}$. Then, we conclude that $F_{\mathcal{X}}(t)\cdot F_{\mathcal{Y}}(t)\leq \min\left(F_{\mathcal{X}}(t),F_{\mathcal{Y}}(t)\right),\ \forall\ t\in\mathbb{R}$ (i.e. $F_{\mathcal{M}\text{ax}_{\text{Indep}}\{\mathcal{X},\mathcal{Y}\}}(t)\leq F_{\mathcal{M}\text{ax}_{\text{Diaz}}\{\mathcal{X},\mathcal{Y}\}}(t),\ \forall\ t\in\mathbb{R}$). Hence, we get $\mathcal{M}\text{ax}_{\text{Indep}}\{\mathcal{X},\mathcal{Y}\}\succeq\mathcal{M}\text{ax}_{\text{Diaz}}\{\mathcal{X},\mathcal{Y}\}$ ■

Probabilistic maximum based on the Fréchet-Hoeffding copula bound

Another way to define an upper bound (maximum) of random variables, is by using the Fréchet-Hoeffding copula bound [3].

Definition 4.10 (From Bernat et al. [3])**.** *Let* $\mathcal{X}$ *and* $\mathcal{Y}$ *be two discrete random variables and* $F_{\mathcal{XY}}$ *is their joint CDF function. Then, we have:*

$$F_{\mathcal{XY}}(x,y)\geq\max\left(F_{\mathcal{X}}(x)+F_{\mathcal{Y}}(y)-1,0\right),\ \forall\ x\in\mathbb{R}\ \textit{and}\ \forall\ y\in\mathbb{R}$$

Definition 4.11. *Let* $\mathcal{X}$ *and* $\mathcal{Y}$ *be two discrete random variables. We denote their maximum, based Fréchet-Hoeffding copula bound (Definition 4.10), by the random variable* $\mathcal{Z}=\mathcal{M}\text{ax}_{\text{Copula}}\{\mathcal{X},\mathcal{Y}\}$. $\mathcal{Z}$ *is defined with its CDF function given below:*

$$F_{\mathcal{Z}}(t)=\max\left(F_{\mathcal{X}}(t)+F_{\mathcal{Y}}(t)-1,0\right)=\left(F_{\mathcal{X}}(t)+F_{\mathcal{Y}}(t)-1\right)^{+}$$

For a given $t \in \mathbb{R}$, if $F_{\mathcal{Z}}(t) = F_{\mathcal{X}}(t) + F_{\mathcal{Y}}(t) - 1$ then $F_{\mathcal{Z}}(t) \leq F_{\mathcal{X}}(t)$ because $F_{\mathcal{Y}}(t) - 1 \leq 0$ (since $0 \leq F_{\mathcal{Y}}(t) \leq 1,\ \forall\ t \in \mathbb{R}$). Otherwise, if $F_{\mathcal{Z}}(t) = 0$ it is obvious that $F_{\mathcal{Z}}(t) \leq F_{\mathcal{X}}(t)$ (since $0 \leq F_{\mathcal{X}}(t) \leq 1,\ \forall\ t \in \mathbb{R}$).

In conclusion, $F_{\mathcal{Z}}(t) \leq F_{\mathcal{X}}(t),\ \forall\ t \in \mathbb{R}$. Therefore, $\mathcal{Z} \succeq \mathcal{X}$. Similarly, we deduce that $\mathcal{Z} \succeq \mathcal{Y}$.

Theorem 4.1. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two discrete and independent random variables and $\mathcal{Z} = \mathcal{M}\mathrm{ax}_{\mathrm{Indep}}\{\mathcal{X}, \mathcal{Y}\}$) Then:*

$$\mathcal{M}\mathrm{ax}_{\mathrm{Copula}}\{\mathcal{X}, \mathcal{Y}\} \succeq \mathcal{Z} \succeq \mathcal{M}\mathrm{ax}_{\mathrm{Diaz}}\{\mathcal{X}, \mathcal{Y}\} \tag{4.3}$$

Proof. The cumulative distribution function $F_{\mathcal{Z}}$ of $\mathcal{Z}$ is computed, $\forall\ t \in \mathbb{R}$, as follows:

$$\begin{aligned} F_{\mathcal{Z}}(t) = P(\mathcal{Z} \leqslant t) &= P(\max(\mathcal{X}, \mathcal{Y}) \leqslant t) \\ &= P(\mathcal{X} \leqslant t, \mathcal{Y} \leqslant t) \\ &= F_{\mathcal{X}\mathcal{Y}}(t, t) \\ &\geq \max(F_{\mathcal{X}}(t) + F_{\mathcal{Y}}(t) - 1, 0) \quad \text{(Fréchet bound Definition 4.10)} \\ &\geq F_{\mathcal{M}\mathrm{ax}_{\mathrm{Copula}}\{\mathcal{X},\mathcal{Y}\}}(t),\ \forall\ t \in \mathbb{R} \end{aligned}$$

Then, we get $\mathcal{M}\mathrm{ax}_{\mathrm{Copula}}\{\mathcal{X}, \mathcal{Y}\} \succeq \mathcal{Z}$

From Lemma 4.1, we deduce that $\mathcal{Z} \succeq \mathcal{M}\mathrm{ax}_{\mathrm{Diaz}}\{\mathcal{X}, \mathcal{Y}\}$

■

Example 4.4. *For instance, based on the Fréchet-Hoeffding copula bound (Definition 4.10), the maximum of probability distributions is defined as follows:*

$$\mathcal{M}\mathrm{ax}_{\mathrm{Copula}}\{\mathcal{X}, \mathcal{Y}\} = \mathcal{M}\mathrm{ax}_{\mathrm{Copula}}\left\{\begin{pmatrix} 3 & 7 \\ 0.3 & 0.7 \end{pmatrix}, \begin{pmatrix} 0 & 4 \\ 0.1 & 0.9 \end{pmatrix}\right\} = \begin{pmatrix} 4 & 7 \\ 0.3 & 0.7 \end{pmatrix}$$

In fact, the corresponding CDF functions are $\begin{pmatrix} 0 & 3 & 4 & 7 \\ 0 & 0.3 & 0.3 & 1 \end{pmatrix}$ *and* $\begin{pmatrix} 0 & 3 & 4 & 7 \\ 0.1 & 0.1 & 1 & 1 \end{pmatrix}$. *Thus, the CDF function of their maximum bound is:*

$$\begin{pmatrix} 0 & 3 & 4 & 7 \\ (0+0.1-1)^+ & (0.3+0.1-1)^+ & (0.3+1-1)^+ & (1+1-1)^+ \end{pmatrix} = \begin{pmatrix} 0 & 3 & 4 & 7 \\ 0 & 0 & 0.3 & 1 \end{pmatrix}$$

From Figure 4.2, we note that the CDF of $\mathcal{M}\mathrm{ax}_{\mathrm{Copula}}\{\mathcal{X}, \mathcal{Y}\}$ (the solid pink line) is below the CDF of $\mathcal{M}\mathrm{ax}_{\mathrm{Indep}}\{\mathcal{X}, \mathcal{Y}\}$ (the solid green line) which is below the CDF of $\mathcal{M}\mathrm{ax}_{\mathrm{Diaz}}\{\mathcal{X}, \mathcal{Y}\}$ (the red dashed line). Hence, we deduce that $\mathcal{M}\mathrm{ax}_{\mathrm{Copula}}\{\mathcal{X}, \mathcal{Y}\} \succeq \mathcal{M}\mathrm{ax}_{\mathrm{Indep}}\{\mathcal{X}, \mathcal{Y}\} \succeq \mathcal{M}\mathrm{ax}_{\mathrm{Diaz}}\{\mathcal{X}, \mathcal{Y}\}$. This result was proven in general by Theorem 4.1.

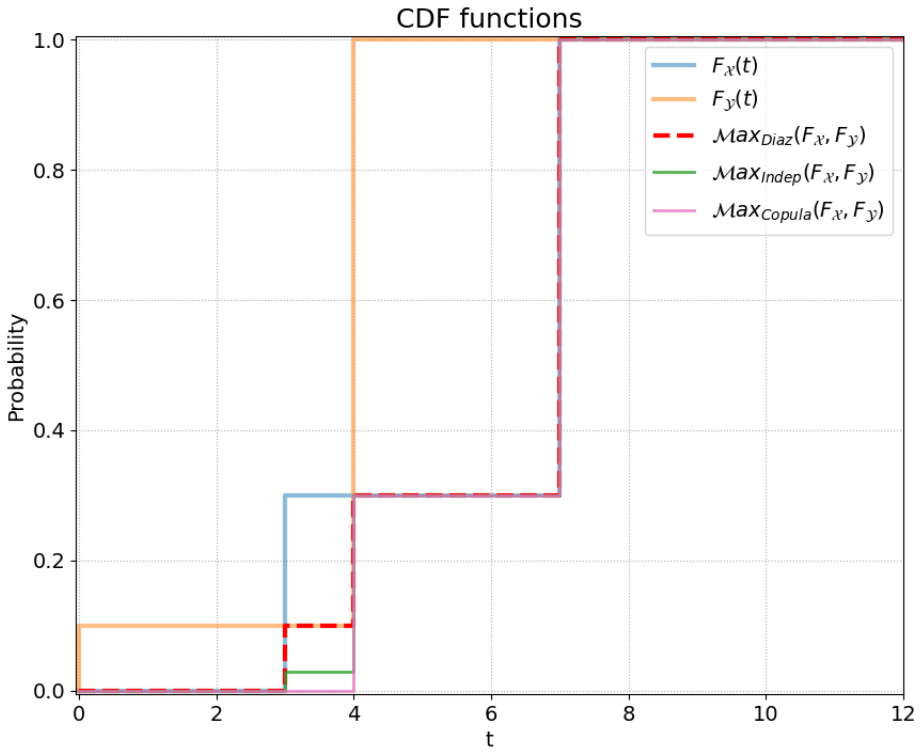


Figure 4.2: CDF functions of distributions examples $\mathcal{X}$ and $\mathcal{Y}$ and their maximums corresponding to different Definitions 4.8, 4.9 and 4.11

Remark. *Equation 4.3 is similar to the Fréchet-Hoeffding copula lower and upper bounds (presented by Equation 4 in Bernat et al. [3]). The only difference is the comparison signs. They are inverted because we use a comparison relation between distributions (Definition 4.3) which is different from the usual comparison between reals.*

4.2.2 Extension of first method equations

After defining probabilistic operators, we use them to extend deterministic equations of Section 3.2.3.1 and to provide a RTA for task model with probabilistic parameters. First, we proceed by replacing the sum and maximum operators, in deterministic response time equations, by the corresponding probabilistic operators (i.e. convolution and probabilistic maximum respectively). Second, we describe how the iterative update of external interference Equation 3.24 (on page 49) is implemented in the case of a probabilistic task set. We also adjust the stop condition to be adapted to probabilistic parameters.

4.2.2.1 Replacing sum and maximum operators

In equations 3.21 and 3.22, we replace the sum operator by the convolution to handle probability distributions instead of scalar values. We also replace the maximum operator by a probabilistic one.

Then, Equation 3.21 becomes

$$\mathcal{R}_{i,j}^{pred} = \underset{\tau_{i,k} \in ipred(\tau_{i,j})}{\mathcal{M}\text{ax}} \left\{ \mathcal{R}_{i,k}^{pred} \otimes \mathcal{E}_i(k,j) \otimes \mathcal{I}_{i,j}^{pred}(\tau_{i,k}) \right\} \otimes \mathcal{C}_{i,j} \tag{4.4}$$

The $\mathcal{M}$ax operator, used in the previous equation, refers to one of the defined probabilistic maximum operators ($\mathcal{M}\text{ax}_{\text{Indep}}$, $\mathcal{M}\text{ax}_{\text{Diaz}}$ or $\mathcal{M}\text{ax}_{\text{Copula}}$). Moreover, $\mathcal{I}_{i,j}^{pred}(\tau_{i,k})$ represents the internal interference caused by predecessors of $\tau_{i,j}$ on $\tau_{i,k}$ and its predecessors. It is defined as follows, similarly to Lemma 3.3 (on page 42):

$$\mathcal{I}_{i,j}^{pred}(\tau_{i,k}) = \bigotimes_{\tau_{i,l} \in \Psi_{i,j}(\tau_{i,k})} \mathcal{C}_{i,l} \tag{4.5}$$

We denote by $\Psi_{i,j}(\tau_{i,k})$ the set composed of predecessor sub-tasks to $\tau_{i,j}$ but not predecessors to $\tau_{i,k}$ that could preempt one of the sub-tasks in $pred^{\star}(\tau_{i,k})$ containing $\tau_{i,k}$ and all its predecessors (see Equation 3.17 on page 43).

Equation 3.22 becomes:

$$\mathcal{R}_{i,j}^{isol} = \mathcal{R}_{i,j}^{pred} \otimes \bigotimes_{\tau_{i,k} \in \Pi_{i,j}^{pred}} \mathcal{C}_{i,k} \tag{4.6}$$

We denote by $\Pi_{i,j}^{pred}$ the set of sub-tasks not predecessors to $\tau_{i,j}$ that could preempt $\tau_{i,j}$ or one of its predecessors. It is defined by Formula 3.20 (on page 44).

Theorem 4.2. *For a probabilistic DAG task set, if all random variables $\mathcal{C}_{i,j}$ and $\mathcal{E}_i(j,k)$ (all pWCETs and pWCCTs) are independent between each other, then the convoluted random variables in response time equations (Equations 4.4, 4.5 and 4.6) are also independent.*

Proof. We assume that all pWCETs and pWCCTs ($\mathcal{C}_{i,j}$ and $\mathcal{E}_i(j,k)$) are independent. Hence, $\mathcal{I}_{i,j}^{pred}(\tau_{i,k})$ (Equation 4.5) equals the convolution of independent random variables and it depends on the pWCET of predecessors of $\tau_{i,j}$ that are not predecessors of $\tau_{i,k}$. In the other hand, $\mathcal{R}_{i,k}^{pred}$ (defined in Equation 4.4) depends only on the pWCET of $\tau_{i,k}$ and its predecessors and the pWCCTs between them. Then, there is no common random variable that the two random variables $\mathcal{R}_{i,k}^{pred}$ and $\mathcal{I}_{i,j}^{pred}(\tau_{i,k})$ depend on. Thus, convolution operators in Equation 4.4 are applied between independent random variables.

For Equation 4.6, the second term is the convolution of independent random variables $\mathcal{C}_{i,k}$. It depends on the pWCET of parallel sub-tasks to $\tau_{i,j}$ and on its predecessors (the set $\Pi_{i,j}^{pred}$). However, the first term $\mathcal{R}_{i,j}^{pred}$ (the preceding response time) does not depend on parallel sub-tasks, it depends only on the pWCET of $\tau_{i,j}$ and its predecessors and the pWCCTs between them. Hence, the convolution operators in Equation 4.6 are used on independent random variables.

■

From Theorem 4.2, we conclude that the convolution operator in response time equations (Equations 4.4, 4.5 and 4.6) is applied on independent random variables as required by the definition of this operator (Definition 4.7).

Definition 4.12. *A DAG is said to have a **polytree** structure if its underlying undirected graph is a connected graph without cycles (acyclic).*

*A DAG is said to be an **arborescence** if it is a rooted polytree, i.e. a polytree with a single source node.*

Theorem 4.3. *The preceding response times $\mathcal{R}_{i,k}^{pred},\ k \in ipred(\tau_{i,j})$ inside the probabilistic maximum operator in Equation 4.4 are independent random variables if the DAG task τ_i has a polytree structure.*

Proof. In Equation 4.4, the preceding response times $\mathcal{R}_{i,k}^{pred}$ of each immediate predecessor $\tau_{i,k}$ depends only on its pWCET $\mathcal{C}_{i,k}$, the pWCETs of its predecessors and the pWCCTs between them.

If two immediate predecessors of $\tau_{i,j}$ have a common predecessor then they will form an undirected cycle with $\tau_{i,j}$ and the common predecessor. This is not possible because the DAG has polytree structure. Thus, immediate predecessors of $\tau_{i,j}$ cannot have a common predecessor and their preceding response times $\mathcal{R}_{i,k}^{pred},\ k \in ipred(\tau_{i,j})$ are independent random variables.

■

The probabilistic maximum operator between independent random variables (Definition 4.8) could be used, in equation 4.4, for some specific cases and DAG structures such as polytree. In general, we should use the maximum operator based on copula bound (Definition 4.11) in order to avoid under-estimating the maximum distribution and to guarantee a safe approximation. However, this maximum operator may provide a pessimistic estimation compared to the exact maximum distribution as noted in the following Example 4.5.

Example 4.5. *In this example, we use equations 4.4, 4.5 and 4.6 to compute response time distributions of the task set described by Figure 4.3 and Table 4.1. We use different definitions of maximum operator and we compare the results obtained with the exact distribution obtained by exploring all possible combinations.*

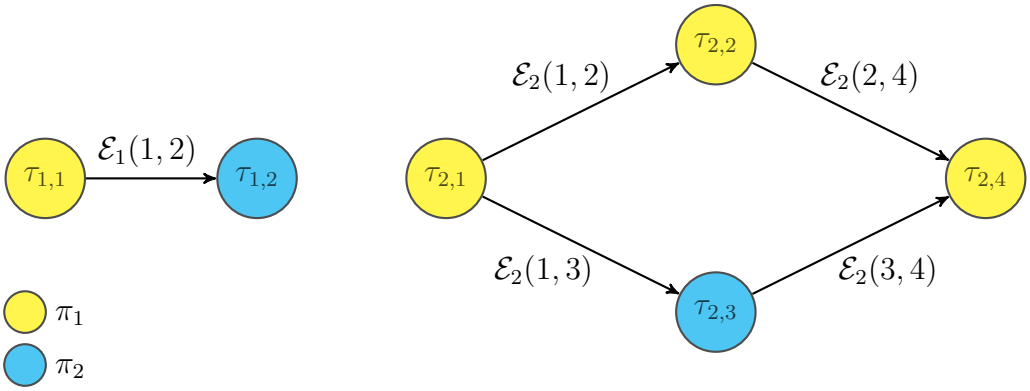


Figure 4.3: Example of a DAG task set with parallel sub-tasks

Table 4.1: Parameters of the task set described in Figure 4.3

Sub-task	$\mathcal{C}_{i,j}$	T_i	Priority	Precedence	delay
$\tau_{1,1}$	$\begin{pmatrix}1\\1\end{pmatrix}$	20	high	$\mathcal{E}_1(1,2)$	$\begin{pmatrix}1\\1\end{pmatrix}$
$\tau_{1,2}$	$\begin{pmatrix}1 & 2\\0.5 & 0.5\end{pmatrix}$	20	high	–	–
$\tau_{2,1}$	$\begin{pmatrix}1 & 5\\0.3 & 0.7\end{pmatrix}$	30	low	$\mathcal{E}_2(1,2)$	$\begin{pmatrix}0\\1\end{pmatrix}$
$\tau_{2,2}$	$\begin{pmatrix}3 & 7\\0.1 & 0.9\end{pmatrix}$	30	low	$\mathcal{E}_2(1,3)$	$\begin{pmatrix}1\\1\end{pmatrix}$
$\tau_{2,3}$	$\begin{pmatrix}4 & 8\\0.6 & 0.4\end{pmatrix}$	30	low	$\mathcal{E}_2(2,4)$	$\begin{pmatrix}0\\1\end{pmatrix}$
$\tau_{2,4}$	$\begin{pmatrix}2\\1\end{pmatrix}$	30	low	$\mathcal{E}_2(3,4)$	$\begin{pmatrix}1\\1\end{pmatrix}$

In Table 4.2, we present the distribution of response times in isolation for each sub-task according to different probabilistic maximum operators; the maximum between independent random variable (Definition 4.8), the maximum based on Diaz comparison of CDF functions (Definition 4.9) and the maximum based on copula bound (Definition 4.11).

Table 4.2: Estimation of the response time in isolation of sub-tasks described in Figure 4.3 and Table 4.1 with different maximum Definitions 4.8, 4.9 and 4.11

Sub-task	$\mathcal{R}_{i,j}^{isol}$(**Indep**)	$\mathcal{R}_{i,j}^{isol}$(**Diaz**)	$\mathcal{R}_{i,j}^{isol}$(**Copula**)
$\tau_{\mathbf{1,1}}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$
$\tau_{\mathbf{1,2}}$	$\begin{pmatrix} 3 & 4 \\ .5 & .5 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \\ .5 & .5 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \\ .5 & .5 \end{pmatrix}$
$\tau_{\mathbf{2,1}}$	$\begin{pmatrix} 1 & 5 \\ .3 & .7 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 \\ .3 & .7 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 \\ .3 & .7 \end{pmatrix}$
$\tau_{\mathbf{2,2}}$	$\begin{pmatrix} 4 & 8 & 12 \\ .03 & .34 & .63 \end{pmatrix}$	$\begin{pmatrix} 4 & 8 & 12 \\ .03 & .34 & .63 \end{pmatrix}$	$\begin{pmatrix} 4 & 8 & 12 \\ .03 & .34 & .63 \end{pmatrix}$
$\tau_{\mathbf{2,3}}$	$\begin{pmatrix} 6 & 10 & 14 \\ .18 & .54 & .28 \end{pmatrix}$	$\begin{pmatrix} 6 & 10 & 14 \\ .18 & .54 & .28 \end{pmatrix}$	$\begin{pmatrix} 6 & 10 & 14 \\ .18 & .54 & .28 \end{pmatrix}$
$\tau_{\mathbf{2,4}}$	$\begin{pmatrix} 9 & 10 & 13 & 14 & 17 \\ .0054 & .0612 & .1998 & .4536 & .28 \end{pmatrix}$	$\begin{pmatrix} 9 & 10 & 13 & 14 & 17 \\ .03 & .15 & .19 & .35 & .28 \end{pmatrix}$	$\begin{pmatrix} 13 & 14 & 17 \\ .09 & .63 & .28 \end{pmatrix}$

For all sub-tasks except for $\tau_{2,4}$, in Table 4.2, we note that the response time distributions are equal for different maximum definitions. Indeed, these sub-tasks have only one immediate predecessor. Thus, the maximum operator in Equation 4.4 yields the input distribution, which is the same for different maximum definitions. However, sub-task $\tau_{2,4}$ has two immediate predecessors. Therefore, the maximum operator is applied on two different distributions. Hence, we get different maximum distributions and different response times according to the maximum operator used.

In Table 4.3, we enumerate all possible combinations of execution time values for all sub-tasks based on their execution time distribution. Each row presents a possible combination with the selected value for each execution time and the corresponding probability. It also contains the computed response time of each sub-task obtained by applying deterministic analysis on selected execution times values.

In Table 4.4, we compute the distribution of response times in isolation for each sub-task from Table 4.3 by summing the probabilities of the concerned combinations for each value in the response time distribution.

For all sub-tasks except for $\tau_{2,4}$, we note that the response time distributions obtained are equal to the ones computed with response time equations (in Table 4.2). In fact, the maximum operator in response time equations is not actually applied for these sub-tasks because they have only one immediate predecessor. However, the only probabilistic operator actually used for these sub-tasks is the convolution. As we proved in Theorem 4.2, convolution in response time equations operates on independent random variables. Therefore, response time distributions computed with these equations are equal to the exact distributions obtained by exploring all combinations.

On the other hand, for sub-task $\tau_{2,4}$, the maximum operator is applied on two dependent distributions because $\tau_{2,4}$ has two immediate predecessors $\tau_{2,2}$ and $\tau_{2,3}$ that

Table 4.3: Deterministic analysis of deterministic task sets obtained by all possible combinations of execution times values.

N°	$\mathbf{C_{1,2}}$	$\mathbf{C_{2,1}}$	$\mathbf{C_{2,2}}$	$\mathbf{C_{2,3}}$	Prob.	$\mathbf{R^{isol}_{1,1}}$	$\mathbf{R^{isol}_{1,2}}$	$\mathbf{R^{isol}_{2,1}}$	$\mathbf{R^{isol}_{2,2}}$	$\mathbf{R^{isol}_{2,3}}$	$\mathbf{R^{isol}_{2,4}}$
1	1	1	3	4	0.009	1	3	1	4	6	9
2	2	1	3	4	0.009	1	4	1	4	6	9
3	1	5	3	4	0.021	1	3	5	8	10	13
4	2	5	3	4	0.021	1	4	5	8	10	13
5	1	1	7	4	0.081	1	3	1	8	6	10
6	2	1	7	4	0.081	1	4	1	8	6	10
7	1	5	7	4	0.189	1	3	5	12	10	14
8	2	5	7	4	0.189	1	4	5	12	10	14
9	1	1	3	8	0.006	1	3	1	4	10	13
10	2	1	3	8	0.006	1	4	1	4	10	13
11	1	5	3	8	0.014	1	3	5	8	14	17
12	2	5	3	8	0.014	1	4	5	8	14	17
13	1	1	7	8	0.054	1	3	1	8	10	13
14	2	1	7	8	0.054	1	4	1	8	10	13
15	1	5	7	8	0.126	1	3	5	12	14	17
16	2	5	7	8	0.126	1	4	5	12	14	17

Table 4.4: Distribution of response times in isolation for all sub-tasks based on exploring all combinations in Table 4.3.

Sub-task	$\mathcal{R}^{isol}_{i,j}$(**All combination**)
$\tau_{\mathbf{1,1}}$	$\left(\begin{smallmatrix}1\\ 2\times.009+2\times.021+2\times.081+2\times.189+2\times.006+2\times.014+2\times.054+2\times.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}1\\ 1\end{smallmatrix}\right)$
$\tau_{\mathbf{1,2}}$	$\left(\begin{smallmatrix}3\\ .009+.021+.081+.189+.006+.014+.054+.126\end{smallmatrix}\ \begin{smallmatrix}4\\ .5\end{smallmatrix}\right) = \left(\begin{smallmatrix}3\\ .5\end{smallmatrix}\ \begin{smallmatrix}4\\ .5\end{smallmatrix}\right)$
$\tau_{\mathbf{2,1}}$	$\left(\begin{smallmatrix}1\\ 2\times.009+2\times.081+2\times.006+2\times.054\end{smallmatrix}\ \begin{smallmatrix}5\\ 2\times.021+2\times.189+2\times.014+2\times.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}1\\ .3\end{smallmatrix}\ \begin{smallmatrix}5\\ .7\end{smallmatrix}\right)$
$\tau_{\mathbf{2,2}}$	$\left(\begin{smallmatrix}4\\ 2\times.009+2\times.006\end{smallmatrix}\ \begin{smallmatrix}8\\ .2\times.021+2\times.081+2\times.014+2\times.054\end{smallmatrix}\ \begin{smallmatrix}12\\ 2\times.189+2\times.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}4\\ .03\end{smallmatrix}\ \begin{smallmatrix}8\\ .34\end{smallmatrix}\ \begin{smallmatrix}12\\ .63\end{smallmatrix}\right)$
$\tau_{\mathbf{2,3}}$	$\left(\begin{smallmatrix}6\\ 2\times.009+2\times.081\end{smallmatrix}\ \begin{smallmatrix}10\\ 2\times.021+2\times.189+2\times.006+2\times.054\end{smallmatrix}\ \begin{smallmatrix}14\\ 2\times.014+2\times.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}6\\ .18\end{smallmatrix}\ \begin{smallmatrix}10\\ .54\end{smallmatrix}\ \begin{smallmatrix}14\\ .28\end{smallmatrix}\right)$
$\tau_{\mathbf{2,4}}$	$\left(\begin{smallmatrix}9\\ 2\times.009\end{smallmatrix}\ \begin{smallmatrix}10\\ 2\times.081\end{smallmatrix}\ \begin{smallmatrix}13\\ 2\times.021+2\times.006+2\times.054\end{smallmatrix}\ \begin{smallmatrix}14\\ 2\times.189\end{smallmatrix}\ \begin{smallmatrix}17\\ 2\times.014+2\times.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}9\\ .018\end{smallmatrix}\ \begin{smallmatrix}10\\ .162\end{smallmatrix}\ \begin{smallmatrix}13\\ .162\end{smallmatrix}\ \begin{smallmatrix}14\\ .378\end{smallmatrix}\ \begin{smallmatrix}17\\ .28\end{smallmatrix}\right)$

have a common predecessor $\tau_{2,1}$. Since the proposed maximum operators are not able to capture these dependencies, they produce an estimate of the maximum distribution that may under-estimate or over-estimate the exact distribution obtained by exploring all combinations. Therefore, in Figure 4.4, we compare the CDF function of exact response time distribution of $\tau_{2,4}$ with CDF functions of response times obtained according to different maximum definitions.

From Figure 4.4, we note that the CDF of the response time in isolation obtained with the copula maximum operator is below the one obtained with independent maximum operator which is also below the CDF of the exact response time distribution

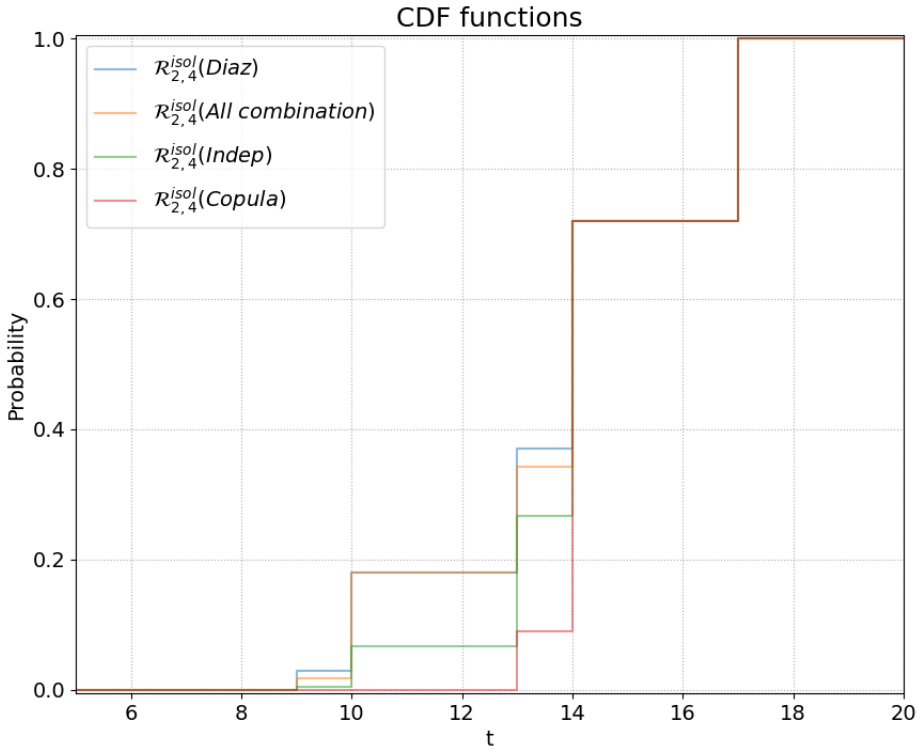


Figure 4.4: Comparison between CDF functions of response time in isolation $\mathcal{R}^{isol}_{2,4}$ computed with different probabilistic maximum Definitions 4.8, 4.9 and 4.11 and the exact one derived from all combinations

derived from all combinations. We also remark that the CDF of the response time in isolation obtained with the Diaz maximum operator is above all other CDF functions. We deduce that $\mathcal{R}^{isol}_{2,4}(Copula) \succeq \mathcal{R}^{isol}_{2,4}(Indep) \succeq \mathcal{R}^{isol}_{2,4}(Combin) \succeq \mathcal{R}^{isol}_{2,4}(Diaz)$. *These comparisons are coherent with the results obtained by Theorem 4.1.*

It is worth noting that the response time obtained with the independent maximum operator is a safe estimate because it does not under-estimate the exact response time distribution (the CDF function never goes above it). It is also less pessimistic than the one obtained with copula maximum because $\mathcal{R}^{isol}_{2,4}(Copula) \succeq \mathcal{R}^{isol}_{2,4}(Indep)$. *Hence, the independent maximum operator allows us to reduce pessimism while guaranteeing a safe estimation of the response time.*

4.2.2.2 Iterative equation for global response time

After estimating the distribution of response times in isolation, we use an iterative equation to compute the global response time distribution. Indeed, we extend Equation 3.23 in order to handle probabilistic parameters. Hence, we replace the usual operators by probabilistic ones like convolution and maximum. Thus,

Equation 3.23 becomes:

$$\mathcal{R}_{i,j}^{global} = \mathcal{R}_{i,j}^{isol} \otimes \bigotimes_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) \in \pi(pred^{\star}(\tau_{i,j}))}} \left\lceil \frac{\mathcal{J}_{p,q} \otimes \mathcal{R}_{i,j}^{global}}{T_p} \right\rceil \mathcal{C}_{p,q} \tag{4.7}$$

The jitter $\mathcal{J}_{i,j}$ of $\tau_{i,j}$ is given by:

$$\mathcal{J}_{i,j} = \underset{\tau_{i,k} \in ipred(\tau_{i,j})}{\mathcal{M}\text{ax}} \left\{ \mathcal{R}_{i,k}^{global} \otimes \mathcal{E}_i(k,j) \right\} \tag{4.8}$$

In the probabilistic case, when considering a preemption from a higher priority task, we should know its activation time because values in the response time distribution that are prior to preemption activation should not be modified, only greater response time values should be preempted and increased. Therefore, we use an algorithm similar to the one proposed by Maxim and Cucu [74] to solve the iterative Equation 4.7.

Algorithm 1: Computation of global response time distribution of $\tau_{i,j}$ according to iterative Equation 4.7

Data: $Preempt(\tau_{i,j})$ set of sub-tasks from higher priority DAG that could preempt $\tau_{i,j}$ or one of its predecessors. $\mathcal{R}_{i,j}^{isol}$ distribution of response time in isolation of $\tau_{i,j}$

Result: $\mathcal{R}_{i,j}^{global}$ global response time distribution of $\tau_{i,j}$

1 $\mathcal{R}_{i,j}^{global} = \mathcal{R}_{i,j}^{isol}$ `/* initialize global response time distribution */`
2 **for** $\tau_{a,b} \in Preempt(\tau_{i,j})$ **do**
3 | $A_{a,b} = 0$ `/* initialize activation time of` $\tau_{a,b}$ `*/`
4 **end**
5 $\tau_{p,q} = \text{argmin}_{\tau_{a,b} \in Preempt(\tau_{i,j})}(A_{a,b})$
6 Next_Activation $= A_{p,q} \ominus \mathcal{J}_{p,q}$
7 **while** $Next\_Activation < max_{value}(\mathcal{R}_{i,j}^{global})$ ***and*** $Next\_Activation < D_i$ **do**
8 | $\mathcal{R}_{i,j}^{global} = doPreemption(\mathcal{R}_{i,j}^{global}, Next\_Activation, \mathcal{C}_{p,q})$
9 | $A_{p,q} = A_{p,q} + T_p$
10 | $\tau_{p,q} = \text{argmin}_{\tau_{a,b} \in Preempt(\tau_{i,j})}(A_{a,b})$
11 | Next_Activation $= A_{p,q} \ominus \mathcal{J}_{p,q}$
12 **end**
13 **return** $\mathcal{R}_{i,j}^{global}$

In Algorithm 1, we compute the distribution of global response time by including higher priority preemptions. It consists of tracking the activation time $A_{a,b}$ of each sub-task $\tau_{a,b} \in Preempt(\tau_{i,j})$ belonging to a higher priority DAG task (lines 2 and 3). Then, we select the higher priority sub-task with the earliest activation (lines 5 and 6) and we perform the preemption using $doPreemption()$ routine (line 8),

which is similar to the method described in the work of Diaz et al. [73] (Figure 2). After that, we update the next activation of the preempting sub-task by adding its minimum inter-arrival time (line 9). Finally, we select the next nearest activation of higher priority sub-tasks (lines 10 and 11) and we repeat the same steps until reaching one of the stop conditions (line 7) described later.

The *doPreemption*() routine is defined in the work of Maxim and Cucu [74] (by algorithm 2). It is composed mainly of three steps; First, we split the response time distribution, at the next activation time of the selected higher priority sub-task, into a *head* and *tail* parts. Second, we preempt the *tail* of the distribution by convolving it with the execution time of the higher priority sub-task. Third, we merge the *head* part and the new *tail* part resulting from the convolution in order to obtain the response time distribution after preemption.

The release jitter of sub-task $\tau_{p,q}$ may delay or advance its activation. In order to deal with the worst scenario and the greatest workload caused by higher priority sub-tasks, we assume that the next activation $A_{p,q}$ of $\tau_{p,q}$ occurs at the earliest possible time and we subtract from it the release jitter $\mathcal{J}_{p,q}$ (line 11 of Algorithm 1) using the $\ominus$ operator, which is a complementary operator of the convolution and is defined in Section 4.2.1.1 (on page 61).

The iterative update and preemption of global response time distribution stop if one of two conditions is reached (line 7 of Algorithm 1). First, if the earliest next activation of higher priority sub-tasks is greater than the highest value in the response time distribution $\mathcal{R}_{i,j}^{global}$, then there is no additional preemption that could occur. Therefore, we should stop iterations. Second, when the next activation of higher priority sub-tasks is greater than the deadline of the studied sub-task $\tau_{i,j}$, then even if additional preemptions could occur they would not change the DMP of this sub-task.

In Chapter 6, we study the performance and run-time of our probabilistic analysis based on probabilistic operators and iterative Algorithm 1 and we compare them to those of other probabilistic analyses presented later in this chapter.

Example 4.6. *In this example, we use iterative Equation 4.7 and Algorithm 1 to compute the global response time distribution of the sub-tasks defined in Figure 4.3 and Table 4.1. We also determine the exact response time distribution obtained by exploring all possible combinations.*

In Table 4.5, we compare the global response time distributions computed with different definitions of the maximum operator. Similarly to Table 4.2, we note that for all sub-tasks except for $\tau_{2,4}$, the global response time distributions are the same for all maximum operator definitions. However, for $\tau_{2,4}$, these distributions are

Table 4.5: Estimation of the global response time of sub-tasks described in Figure 4.3 and Table 4.1 with different maximum Definitions 4.8, 4.9 and 4.11

Sub-task	$\mathcal{R}_{i,j}^{glob}$(**Indep**)	$\mathcal{R}_{i,j}^{glob}$(**Diaz**)	$\mathcal{R}_{i,j}^{glob}$(**Copula**)
$\tau_{\mathbf{1},\mathbf{1}}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$
$\tau_{\mathbf{1},\mathbf{2}}$	$\begin{pmatrix} 3 & 4 \\ .5 & .5 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \\ .5 & .5 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \\ .5 & .5 \end{pmatrix}$
$\tau_{\mathbf{2},\mathbf{1}}$	$\begin{pmatrix} 2 & 6 \\ .3 & .7 \end{pmatrix}$	$\begin{pmatrix} 2 & 6 \\ .3 & .7 \end{pmatrix}$	$\begin{pmatrix} 2 & 6 \\ .3 & .7 \end{pmatrix}$
$\tau_{\mathbf{2},\mathbf{2}}$	$\begin{pmatrix} 5 & 9 & 13 \\ .03 & .34 & .63 \end{pmatrix}$	$\begin{pmatrix} 5 & 9 & 13 \\ .03 & .34 & .63 \end{pmatrix}$	$\begin{pmatrix} 5 & 9 & 13 \\ .03 & .34 & .63 \end{pmatrix}$
$\tau_{\mathbf{2},\mathbf{3}}$	$\begin{pmatrix} 8 & 9 & 12 & 13 & 16 & 17 \\ .09 & .09 & .27 & .27 & .14 & .14 \end{pmatrix}$	$\begin{pmatrix} 8 & 9 & 12 & 13 & 16 & 17 \\ .09 & .09 & .27 & .27 & .14 & .14 \end{pmatrix}$	$\begin{pmatrix} 8 & 9 & 12 & 13 & 16 & 17 \\ .09 & .09 & .27 & .27 & .14 & .14 \end{pmatrix}$
$\tau_{\mathbf{2},\mathbf{4}}$	$\begin{pmatrix} 11 & 12 & 13 & 15 & 16 & 17 & 19 & 22 & 23 \\ .0027 & .0333 & .0306 & .0999 & .3267 & .2268 & .14 & .07 & .07 \end{pmatrix}$	$\begin{pmatrix} 11 & 12 & 13 & 15 & 16 & 17 & 19 & 22 & 23 \\ .015 & .09 & .075 & .095 & .27 & .175 & .14 & .07 & .07 \end{pmatrix}$	$\begin{pmatrix} 15 & 16 & 17 & 19 & 22 & 23 \\ .045 & .36 & .315 & .14 & .07 & .07 \end{pmatrix}$

different. This observation stems from the same observation on distributions of response time in isolation that is included in Equation 4.7 for the calculation of the global response time.

To illustrate the stop conditions of Algorithm 1 (line 7), we examine the global response time computation of $\tau_{2,2}$*. After the first preemption of higher-priority sub-task* $\tau_{1,1}$ *at time* 0*, its next activation occurs after its minimum inter-arrival time at instant* 20 *because its jitter* $\mathcal{J}_{1,1}$ *is equal to zero. This next activation is greater than* 13*, the maximum response time value in* $\mathcal{R}_{2,2}^{global}$ *distribution. Therefore,* $\tau_{1,1}$ *cannot preempt* $\tau_{2,2}$ *any more and we should stop iterations.*

Alternatively, we may stop iterations due to the second stop condition. To highlight it, let us assume that the deadline of $\tau_{2,2}$ *is equal to* 10 *(instead of* 30*) and the minimum inter-arrival time of* $\tau_{1,1}$ *is equal to* 12 *(instead of* 20*). In this case, after the first preemption of higher-priority sub-task* $\tau_{1,1}$*, its next activation becomes equal to* 12*, which is lower than* 13 *the maximum value in* $\mathcal{R}_{2,2}$*. Thus, a second preemption could occur. However, the next activation of* $\tau_{1,1}$ *is greater than the deadline of* $\tau_{2,2}$ *(*$12 > 10$*) so there is no need to preform this preemption because the DMP of* $\tau_{2,2}$ *remains unchangeable. It is equal to* 0.63 *whether the second preemption is performed or not.*

In Tables 4.6 and 4.7, we enumerate all possible combinations and we compute the global response time distribution for each sub-task. For lines 12 *and* 16 *in Table 4.6, there are two values for* $\mathcal{R}_{2,4}^{global}$ *because each value depends on the execution time of the second job of* $\tau_{1,2}$ *that causes the second preemption. The column* $C_{1,2}$*, in this table, represents the execution time of the first job only. Hence, in lines* 12 *and* 16*, each response time value of* $\tau_{2,4}$ *has a probability of the corresponding line multiplied by the probability in* $\mathcal{C}_{1,2}$ *distribution.*

Table 4.6: Deterministic global response time computation of deterministic task sets obtained by all possible combinations of execution time values.

N°	$C_{1,2}$	$C_{2,1}$	$C_{2,2}$	$C_{2,3}$	Prob.	$R^{glob}_{1,1}$	$R^{glob}_{1,2}$	$R^{glob}_{2,1}$	$R^{glob}_{2,2}$	$R^{glob}_{2,3}$	$R^{glob}_{2,4}$
1	1	1	3	4	0.009	1	3	2	5	8	11
2	2	1	3	4	0.009	1	4	2	5	9	12
3	1	5	3	4	0.021	1	3	6	9	12	15
4	2	5	3	4	0.021	1	4	6	9	13	16
5	1	1	7	4	0.081	1	3	2	9	8	12
6	2	1	7	4	0.081	1	4	2	9	9	13
7	1	5	7	4	0.189	1	3	6	13	12	16
8	2	5	7	4	0.189	1	4	6	13	13	17
9	1	1	3	8	0.006	1	3	2	5	12	15
10	2	1	3	8	0.006	1	4	2	5	13	16
11	1	5	3	8	0.014	1	3	6	9	16	19
12	2	5	3	8	0.014	1	4	6	9	17	22/23
13	1	1	7	8	0.054	1	3	2	9	12	15
14	2	1	7	8	0.054	1	4	2	9	13	16
15	1	5	7	8	0.126	1	3	6	13	16	19
16	2	5	7	8	0.126	1	4	6	13	17	22/23

Table 4.7: Distribution of global response time for all sub-tasks based on exploring all the combinations in Table 4.6.

Sub-task	$\mathcal{R}^{glob}_{i,j}$(**All combinations**)
$\tau_{1,1}$	$\left(\begin{smallmatrix}1\\ 2\times.009+2\times.021+2\times.081+2\times.189+2\times.006+2\times.014+2\times.054+2\times.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$
$\tau_{1,2}$	$\left(\begin{smallmatrix}3\\ .009+.021+.081+.189+.006+.014+.054+.126\end{smallmatrix}\ \begin{smallmatrix}4\\.5\end{smallmatrix}\right) = \left(\begin{smallmatrix}3\\.5\end{smallmatrix}\ \begin{smallmatrix}4\\.5\end{smallmatrix}\right)$
$\tau_{2,1}$	$\left(\begin{smallmatrix}2\\ 2\times.009+2\times.081+2\times.006+2\times.054\end{smallmatrix}\ \begin{smallmatrix}6\\2\times.021+2\times.189+2\times.014+2\times.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}2\\.3\end{smallmatrix}\ \begin{smallmatrix}6\\.7\end{smallmatrix}\right)$
$\tau_{2,2}$	$\left(\begin{smallmatrix}5\\ 2\times.009+2\times.006\end{smallmatrix}\ \begin{smallmatrix}9\\.2\times.021+2\times.081+2\times.014+2\times.054\end{smallmatrix}\ \begin{smallmatrix}13\\2\times.189+2\times.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}5\\.03\end{smallmatrix}\ \begin{smallmatrix}9\\.34\end{smallmatrix}\ \begin{smallmatrix}13\\.63\end{smallmatrix}\right)$
$\tau_{2,3}$	$\left(\begin{smallmatrix}8\\ .009+.081\end{smallmatrix}\ \begin{smallmatrix}9\\.009+.081\end{smallmatrix}\ \begin{smallmatrix}12\\.021+.189+.006+.054\end{smallmatrix}\ \begin{smallmatrix}13\\.27\end{smallmatrix}\ \begin{smallmatrix}16\\.014+.126\end{smallmatrix}\ \begin{smallmatrix}17\\.014+.126\end{smallmatrix}\right) = \left(\begin{smallmatrix}8\\.09\end{smallmatrix}\ \begin{smallmatrix}9\\.09\end{smallmatrix}\ \begin{smallmatrix}12\\.27\end{smallmatrix}\ \begin{smallmatrix}13\\.27\end{smallmatrix}\ \begin{smallmatrix}16\\.14\end{smallmatrix}\ \begin{smallmatrix}17\\.14\end{smallmatrix}\right)$
$\tau_{2,4}$	$\left(\begin{smallmatrix}11\\ .009\end{smallmatrix}\ \begin{smallmatrix}12\\.009+.081\end{smallmatrix}\ \begin{smallmatrix}13\\.081\end{smallmatrix}\ \begin{smallmatrix}15\\.021+.006+.054\end{smallmatrix}\ \begin{smallmatrix}16\\.021+.189+.006+.054\end{smallmatrix}\ \begin{smallmatrix}17\\.189\end{smallmatrix}\ \begin{smallmatrix}19\\.014+.126\end{smallmatrix}\ \begin{smallmatrix}22\\.007+.063\end{smallmatrix}\ \begin{smallmatrix}23\\.007+.063\end{smallmatrix}\right)$

$$\mathcal{R}^{global}_{2,4}(\text{All combinations}) = \begin{pmatrix} 11 & 12 & 13 & 15 & 16 & 17 & 19 & 22 & 23 \\ .009 & .09 & .081 & .081 & .27 & .189 & .14 & .07 & .07 \end{pmatrix}$$

From Figure 4.5, we have similar observations to those from Figure 4.4. We note that there is a total order between different distributions. Indeed, from the layout of CDF functions, we deduce that $\mathcal{R}^{isol}_{2,4}(Copula) \succeq \mathcal{R}^{isol}_{2,4}(Indep) \succeq \mathcal{R}^{isol}_{2,4}(Combin) \succeq \mathcal{R}^{isol}_{2,4}(Diaz)$ *which is coherent with the results obtained by Theorem 4.1. In addition, the maximum operator between independent random variables provides a safe*

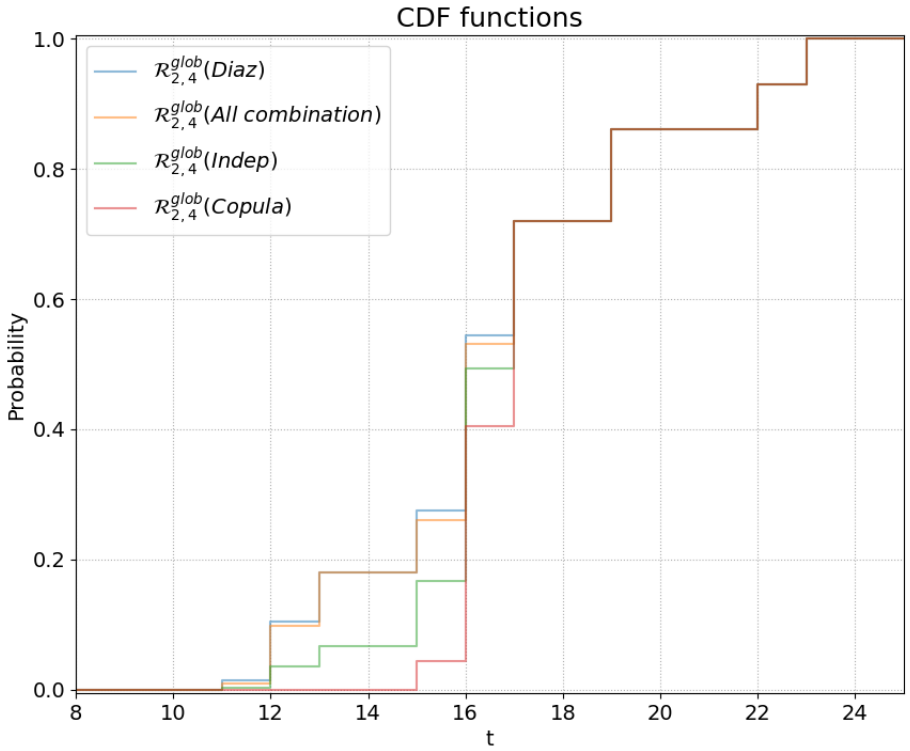


Figure 4.5: Comparison between CDF functions of global response time $\mathcal{R}^{global}_{2,4}$ computed with different probabilistic maximum according to Definitions 4.8, 4.9 and 4.11 and the exact one derived from all combinations

estimate of the exact response time while reducing pessimism compared to the maximum operator based on copula bound.

4.2.3 Extension of second method equations

Following on the previous Section 4.2.2, we use similar techniques like probabilistic operators and iterative preemptions algorithm to extend the deterministic equations of Section 3.2.3.2 in order to analyze the task model with probabilistic parameters.

By replacing the sum and maximum operators by convolution and probabilistic maximum respectively, Equation 3.26 becomes:

$$\mathcal{R}^{pred}_{i,j} = \mathcal{C}_{i,j} \otimes \underset{\tau_{i,k} \in ipred(\tau_{i,j}}{\mathcal{M}\text{ax}} \left\{ \mathcal{R}^{pred}_{i,k} \otimes \mathcal{I}_{i,j}(\tau_{i,k}) \otimes \mathcal{E}_i(k,j) \otimes \mathcal{I}^{pred}_{i,j}(\tau_{i,k}) \right\} \tag{4.9}$$

We denote by $\mathcal{I}_{i,j}(\tau_{i,k})$ the external interference distribution when the immediate predecessor $\tau_{i,k}$ is executed on a different core than $\tau_{i,j}$.

$$\mathcal{I}_{i,j}(\tau_{i,k}) = \begin{cases} \mathcal{I}^{ext}(\tau_{i,k}) & \text{if } \pi(\tau_{i,k}) \neq \pi(\tau_{i,j}) \\ \binom{0}{1} & \text{otherwise} \end{cases}$$

The external interference distribution $\mathcal{I}^{ext}(\tau_{i,j})$ is computed using the following iterative Equation 4.10. This equation is derived from the deterministic Equation 3.27 and it is resolved in a similar way to iterative Equation 4.7 by using Algorithm 1.

$$\mathcal{I}^{ext}(\tau_{i,j}) = \bigotimes_{\substack{\tau_{p,q} \in hep(\tau_{i,j}) \\ \pi(\tau_{p,q}) = \pi(\tau_{i,j})}} \left\lceil \frac{\mathcal{J}_{p,q} \otimes \mathcal{I}^{ext}(\tau_{i,j}) \otimes \mathcal{I}^{cnx}(\tau_{i,j})}{T_p} \right\rceil \mathcal{C}_{p,q} \tag{4.10}$$

The jitter $\mathcal{J}_{p,q}$ of sub-task $\tau_{p,q}$ is computed as in the previous Section 4.2.2 using Equation 4.8.

Equation 3.28 becomes:

$$\mathcal{I}^{cnx}(\tau_{i,j}) = \bigotimes_{\tau_{i,k} \in G^{cnx}_{pred}(\tau_{i,j})} \mathcal{C}_{i,k} \otimes \bigotimes_{\tau_{i,k} \in \Pi^{cnx}_{i,j}} \mathcal{C}_{i,k} \tag{4.11}$$

where the sets $G^{cnx}_{pred}(\tau_{i,j})$ and $\Pi^{cnx}_{i,j}$ are defined respectively by Formulas 3.25 and 3.29 (on page 51).

Finally, the computation of the global response time distribution is derived from deterministic Equation 3.30 as follows:

$$\mathcal{R}^{global}_{i,j} = \mathcal{R}^{pred}_{i,j} \otimes \bigotimes_{\tau_{i,k} \in \Pi^{pred}_{i,j}} \mathcal{C}_{i,k} \otimes \mathcal{I}^{ext}(\tau_{i,j}) \tag{4.12}$$

We denote by $\Pi^{pred}_{i,j}$ the set of sub-tasks that are not predecessors of $\tau_{i,j}$ that could preempt $\tau_{i,j}$ or one of its predecessors. It is defined more formally by Equation 3.20 (on page 44).

Example 4.7. *In this example, we illustrate the results of applying previous response time equations on the task set defined by Figure 4.3 and Table 4.1. We also determine the exact response time distribution obtained by exploring all possible combinations.*

In Table 4.8, we compute the preceding response time and the global response time distributions for each sub-task using the maximum operator between independent random variables. We also evaluate the external interference and the jitter distributions.

We focus on the independent maximum operator because it provides a safe estimate of the maximum distribution and it reduces the pessimism. Moreover, we do not compare the results of independent maximum to the results of other maximum operators because we have already studied the order between these operators in Theorem 4.1 and we have illustrated it in Section 4.2.2 (Figures 4.4 and 4.5).

$$\mathcal{R}^{global}_{2,4}(\text{All combinations}) = \begin{pmatrix} 12 & 13 & 16 & 17 & 20 & 21 \\ .09 & .09 & .27 & .27 & .14 & .14 \end{pmatrix}$$

Table 4.8: Computation of preceding and global response times of sub-tasks described in Figure 4.3 and Table 4.1 using Equations 4.9, 4.10 and 4.12

Sub-task	$\mathcal{R}^{pred}_{i,j}$	$\mathcal{I}^{ext}_{i,j}$	$\mathcal{R}^{glob}_{i,j}$	$\mathcal{J}_{i,j}$
$\tau_{\mathbf{1,1}}$	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$
$\tau_{\mathbf{1,2}}$	$\begin{pmatrix}3&4\\.5&.5\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$	$\begin{pmatrix}3&4\\.5&.5\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$
$\tau_{\mathbf{2,1}}$	$\begin{pmatrix}1&5\\.3&.7\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}2&6\\.3&.7\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$
$\tau_{\mathbf{2,2}}$	$\begin{pmatrix}4&8&12\\.03&.34&.63\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}5&9&13\\.03&.34&.63\end{pmatrix}$	$\begin{pmatrix}2&6\\.3&.7\end{pmatrix}$
$\tau_{\mathbf{2,3}}$	$\begin{pmatrix}7&11&15\\.18&.54&.28\end{pmatrix}$	$\begin{pmatrix}1&2\\.5&.5\end{pmatrix}$	$\begin{pmatrix}8&9&12&13&16&17\\.09&.09&.27&.27&.14&.14\end{pmatrix}$	$\begin{pmatrix}3&7\\.3&.7\end{pmatrix}$
$\tau_{\mathbf{2,4}}$	$\begin{pmatrix}11&12&14&15&16&19&20\\.0333&.0333&.1134&.27&.27&.14&.14\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$	$\begin{pmatrix}12&13&15&16&17&20&21\\.0333&.0333&.1134&.27&.27&.14&.14\end{pmatrix}$	$\begin{pmatrix}9&10&13&14&17&18\\.0333&.0333&.3834&.27&.14&.14\end{pmatrix}$

Table 4.9: Deterministic global response time computation of deterministic task sets obtained by all possible combinations of execution time values.

N°	$\mathbf{C_{1,2}}$	$\mathbf{C_{2,1}}$	$\mathbf{C_{2,2}}$	$\mathbf{C_{2,3}}$	**Prob.**	$\mathbf{R^{glob}_{1,1}}$	$\mathbf{R^{glob}_{1,2}}$	$\mathbf{R^{glob}_{2,1}}$	$\mathbf{R^{glob}_{2,2}}$	$\mathbf{R^{glob}_{2,3}}$	$\mathbf{R^{glob}_{2,4}}$
1	1	1	3	4	0.009	1	3	2	5	8	12
2	2	1	3	4	0.009	1	4	2	5	9	13
3	1	5	3	4	0.021	1	3	6	9	12	16
4	2	5	3	4	0.021	1	4	6	9	13	17
5	1	1	7	4	0.081	1	3	2	9	8	12
6	2	1	7	4	0.081	1	4	2	9	9	13
7	1	5	7	4	0.189	1	3	6	13	12	16
8	2	5	7	4	0.189	1	4	6	13	13	17
9	1	1	3	8	0.006	1	3	2	5	12	16
10	2	1	3	8	0.006	1	4	2	5	13	17
11	1	5	3	8	0.014	1	3	6	9	16	20
12	2	5	3	8	0.014	1	4	6	9	17	21
13	1	1	7	8	0.054	1	3	2	9	12	16
14	2	1	7	8	0.054	1	4	2	9	13	17
15	1	5	7	8	0.126	1	3	6	13	16	20
16	2	5	7	8	0.126	1	4	6	13	17	21

In Tables 4.9 and 4.10, we explore all possible combinations and we compute the global response time distribution for each sub-task. Similarly to the previous Section 4.2.2, we note that, for all sub-tasks except $\tau_{2,4}$, the global response time distributions computed with response time equations are equal to the exact ones derived from exploring all combinations. However, for $\tau_{2,4}$, the maximum operator in Equation 4.9 is applied on two dependent random variables because $\tau_{2,4}$ has two immediate predecessors ($\tau_{2,2}$ and $\tau_{2,3}$) that have a common predecessor ($\tau_{2,1}$). Hence, the computed distribution according to response time equations is different from the

Table 4.10: Distribution of global response times for all sub-tasks based on exploring all the combinations in Table 4.9.

Sub-task	$\mathcal{R}_{i,j}^{glob}$**(All combinations)**
$\tau_{\mathbf{1},\mathbf{1}}$	$\left(\overset{1}{2\times.009+2\times.021+2\times.081+2\times.189+2\times.006+2\times.014+2\times.054+2\times.126}\right) = \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$
$\tau_{\mathbf{1},\mathbf{2}}$	$\left(\overset{3}{.009+.021+.081+.189+.006+.014+.054+.126}\ \overset{4}{.5}\right) = \left(\begin{smallmatrix}3 & 4\\.5 & .5\end{smallmatrix}\right)$
$\tau_{\mathbf{2},\mathbf{1}}$	$\left(\overset{2}{2\times.009+2\times.081+2\times.006+2\times.054}\ \overset{6}{2\times.021+2\times.189+2\times.014+2\times.126}\right) = \left(\begin{smallmatrix}2 & 6\\.3 & .7\end{smallmatrix}\right)$
$\tau_{\mathbf{2},\mathbf{2}}$	$\left(\overset{5}{2\times.009+2\times.006}\ \overset{9}{.2\times.021+2\times.081+2\times.014+2\times.054}\ \overset{13}{2\times.189+2\times.126}\right) = \left(\begin{smallmatrix}5 & 9 & 13\\.03 & .34 & .63\end{smallmatrix}\right)$
$\tau_{\mathbf{2},\mathbf{3}}$	$\left(\overset{8}{.009+.081}\ \overset{9}{.009+.081}\ \overset{12}{.021+.189+.006+.054}\ \overset{13}{.27}\ \overset{16}{.014+.126}\ \overset{17}{.014+.126}\right) = \left(\begin{smallmatrix}8 & 9 & 12 & 13 & 16 & 17\\.09 & .09 & .27 & .27 & .14 & .14\end{smallmatrix}\right)$
$\tau_{\mathbf{2},\mathbf{4}}$	$\left(\overset{12}{.009+.081}\ \overset{13}{.009+.081}\ \overset{16}{.021+.189+.006+.054}\ \overset{17}{.021+.189+.006+.054}\ \overset{20}{.014+.126}\ \overset{21}{.014+.126}\right)$

exact distribution since the proposed maximum operators are not able to capture these dependencies.

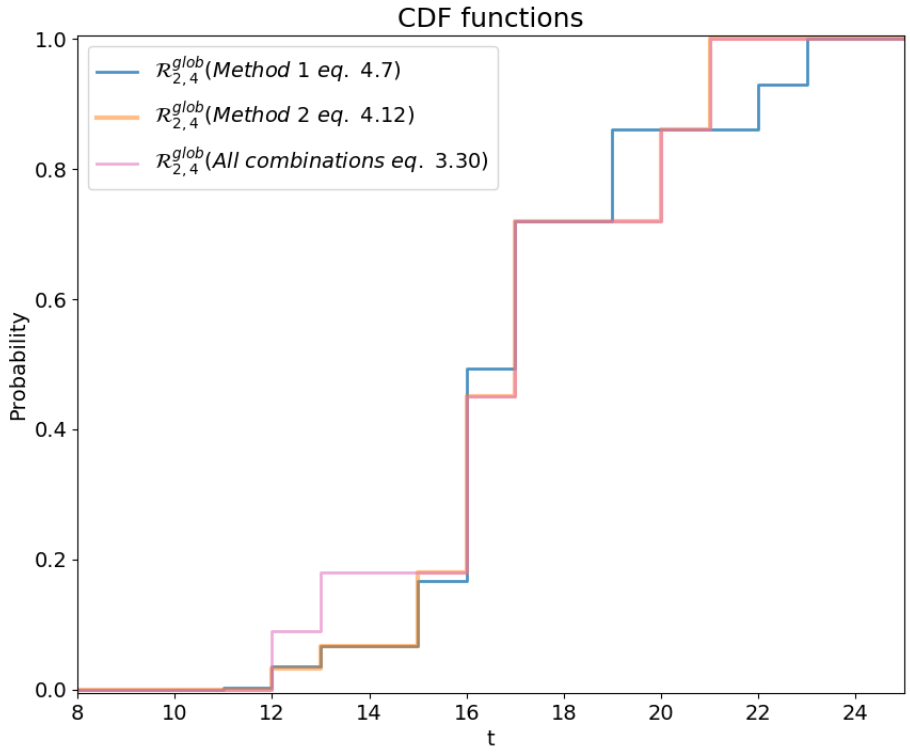


Figure 4.6: Comparison between CDF functions of global response times $\mathcal{R}_{2,4}^{global}$ computed with the response time equations of Section 4.2.2 (referred to as Method 1) and those of Section 4.2.3 (referred to as Method 2) and the exact distribution derived from all combinations

In Figure 4.6, we show that the CDF function of the global response time computed with the response time equations of this Section 4.2.3 (the orange line) is below the exact distribution derived from all combinations (the pink line). Similarly to Figures 4.4 and 4.5, we deduce that $\mathcal{R}_{2,4}^{global}(Indep) \succeq \mathcal{R}_{2,4}^{global}(Combin)$. Hence, the independent maximum operator used in response time equations helps to compute a safe estimate compared to the exact distribution.

In addition, we note that the response time distributions computed with the response time equations of Section 4.2.2 (Method 1) and those of Section 4.2.3 (Method 2) are not comparable because the corresponding CDF functions cross (blue and orange lines). Indeed, $\mathcal{R}_{2,4}^{global}$ obtained with the equations of Method 2 has a smaller worst-case value compared to those obtained using Method 1 since they include only one preemption of $\tau_{1,2}$. This is due to the fact that $\tau_{1,2}$ could preempt $\tau_{2,3}$ only once since its execution in the worst-case finishes at 17 before the second activation of $\tau_{1,2}$ at $19 = 20 - 1$ (minimum inter-arrival time of $\tau_{1,2}$ minus its jitter). However, the equations of Method 1 account for this preemption on the whole DAG. They consider that $\tau_{1,2}$ could preempt the sub-tasks of DAG task τ_2 twice. Thus, they provide a higher response time in the worst case than the equations of Method 2.

4.3 Bayesian Network Inference For Dependent Random Variables

Some hardware architectures and features, like multi-core processors and cache levels, may cause dependency in the execution time of independent software components and programs (sub-tasks). For instance, if two sub-tasks are executed on different cores then the memory access of one may delay the access time and the execution time of the other especially, if they access to the same variable on an architecture with multi-levels of cache memory. Moreover, some structures of precedence constraint between software components may also cause dependency between random variables used in the response time equations of the Section 4.2.

However, the proposed probabilistic operators (like convolution and maximum) used in the response time equations do not deal with dependencies between the random variables involved which may cause under-estimation or over-estimation of the exact response time distribution. Therefore, in this section, we use a Bayesian network to model existing dependencies between random variables used in the response time equations. Then, we use inference techniques used on Bayesian networks to determine the exact distribution of each random variable and of the response time.

4.3.1 Modeling Dependencies

Bayesian networks are used to model dependencies between random variables. They use a DAG graph to describe these dependencies, where nodes represent random variables and edges indicate conditional dependencies between these variables. Each node $\mathcal{X}$ is characterized by a Conditional Probability Table (CPT) that describes the distribution of the concerned node conditioned to its parent. The rows of this table represent all possible combinations of values of $\mathcal{X}$ and of its parent variables. Each of them has a corresponding probability $P(\mathcal{X} \mid parents(\mathcal{X}))$. For example, let $\mathcal{X}$ be a node that has 5 possible values and 3 parent nodes. If each parent variable has two possible values, then the CPT of node $\mathcal{X}$ will have 5×2^3 rows that represent all possible combinations of $\mathcal{X}$ and its parent variables.

For a source node (without any parent), the CPT is equal to the probability distribution of the corresponding random variable. In such a case, the sum of the probabilities of all the rows in the CPT table is equal to 1. For other nodes, the sum of all probabilities is equal to the number of possible combinations taken by parent nodes i.e. the cardinality of parent nodes: $|parents(\mathcal{X})| = \prod_{\mathcal{A} \in parents(\mathcal{X})} K_{\mathcal{A}}$, where $K_{\mathcal{A}}$ is the number of possible values of random variable $\mathcal{A}$ (as introduced in Definition 4.4).

The direction of an edge in the dependency graph may indicate a causality (cause-effect) relation between parents and child nodes. However, in some cases, edges represent influence or correlation between variables and not necessarily a causality relation. For instance, if two random variables (nodes) have a common cause but this cause random variable is not presented in the model, then the values taken by these two nodes are correlated and dependent but the direction of the edge between them it is not clearly defined because there is not a clear cause-effect relation. In general, we say that directed edges encode conditional dependence or independence i.e. a random variable $\mathcal{X}$ in a Bayesian network is independent from other variables in the graph given its parent variables i.e. $P(\mathcal{X} \mid parents(\mathcal{X}), \mathcal{Y}, \mathcal{Z}, \cdots) = P(\mathcal{X} \mid parents(\mathcal{X}))$.

Definition 4.13. *A Bayesian network is defined by a list of* ***random variables*** *(nodes) V, a dependency* ***graph*** *between these random variables and a list of* ***factors*** *(also known as CPT tables) F for all nodes.*

In this section, we study existing dependencies between random variables used in response time Equations 4.4, 4.5 and 4.6. As mentioned in Theorem 4.2 convolution operator is applied between independent random variables in these response time equations. However, the maximum operator in Equation 4.4 is

applied between dependent random variables because the response time of two immediate predecessors could depend on the same random variable (e.g. the response time of a common predecessor). Therefore, we use a Bayesian network to study dependencies between random variables involved in the maximum operator in the preceding response time Equation 4.4.

we recall that the preceding response time of sub-task $\tau_{i,j}$ is given by:

$$\mathcal{R}_{i,j}^{pred} = \underset{\tau_{i,k} \in ipred(\tau_{i,j})}{\mathcal{M}\text{ax}} \left\{ \mathcal{R}_{i,k}^{pred} \otimes \mathcal{E}_i(k,j) \otimes \mathcal{I}_{i,j}^{pred}(\tau_{i,k}) \right\} \otimes \mathcal{C}_{i,j}$$

To distinguish random variables involved in the preceding response time equation, we define $\mathcal{S}_{i,j}(\tau_{i,k})$ as the random variable equal to the sum (convolution) of the three random variables inside the maximum operator of immediate predecessor $\tau_{i,k}$. Then, we write $\mathcal{S}_{i,j}(\tau_{i,k}) = \mathcal{R}_{i,k}^{pred} \otimes \mathcal{E}_i(k,j) \otimes \mathcal{I}_{i,j}^{pred}(\tau_{i,k})$. We also define the random variable $\mathcal{M}_{i,j}$ as the resulting maximum distribution in $\mathcal{R}_{i,j}^{pred}$ equation over all immediate predecessors. Hence, the equation of preceding response time $\mathcal{R}_{i,j}^{pred}$ could be written as follows:

$$\mathcal{R}_{i,j}^{pred} = \underset{\tau_{i,k} \in ipred(\tau_{i,j})}{\mathcal{M}\text{ax}} \{ \mathcal{S}_{i,j}(\tau_{i,k}) \} \otimes \mathcal{C}_{i,j} = \mathcal{M}_{i,j} \otimes \mathcal{C}_{i,j}$$

Example 4.8. *For DAG task τ_2 in Figure 4.3, the Bayesian graph that describes dependencies between random variables involved in the preceding response time Equation 4.4 of sub-task $\tau_{2,4}$ is given below:*

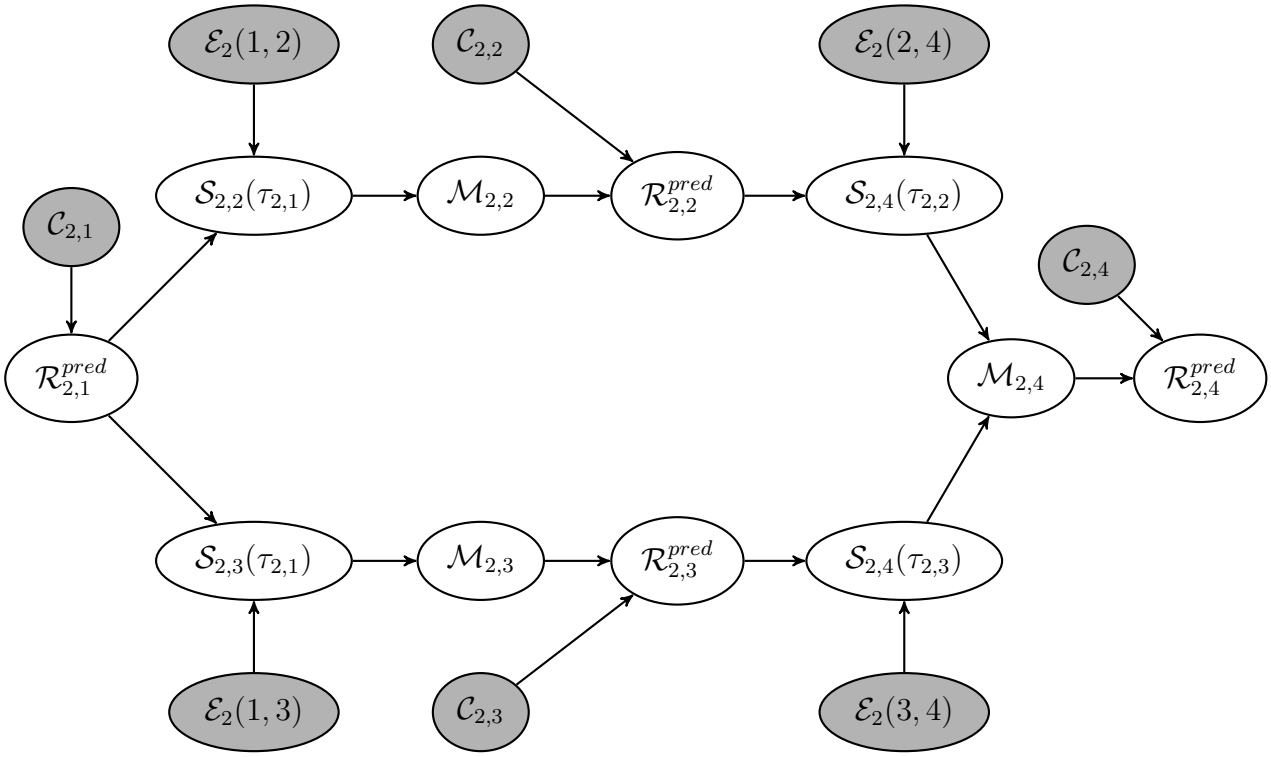


Figure 4.7: Dependency graph between random variables involved in $R_{2,4}^{pred}$ equation.

We note that the Bayesian graph of dependencies between random variables is an acyclic graph with directed edges (DAG). The gray nodes represent random variables corresponding to the timing parameter of the task set (pWCETs and pWCCTs).

Remark. *In this example, we assume that pWCETs and pWCCTs are independent hence they have not any incoming edges. However, in general, they could be dependent and we should add required edges and adapt CPT tables to model these dependencies.*

CPT tables (factors) of gray nodes are equal to the probability distribution of their corresponding random variables. For instance, Tables 4.11 and 4.12 correspond to the probability distribution of random variables $\mathcal{C}_{2,1}$ *and* $\mathcal{E}_2(1,3)$ *respectively, given in Table 4.1. We also note that the probabilities of their rows sum to 1 since they represent probability distributions of a random variable.*

Other CPT tables (from Table 4.13 to Table 4.23) describe the probability distribution of each node conditioned to its parents. For instance, Table 4.15 represents the probability distribution of $\mathcal{S}_{2,3}(\tau_{2,1})$ *given its parents;* $\mathcal{R}_{2,1}^{pred}$ *and* $\mathcal{E}_2(1,3)$*. The number of rows in this table is equal to the product of the number of possible values for each random variable involved in the conditional distribution (i.e.* $K_{\mathcal{S}_{2,3}(\tau_{2,1})} \times K_{\mathcal{R}_{2,1}^{pred}} \times K_{\mathcal{E}_2(1,3)} = 2 \times 2 \times 1 = 4$*). The sum of probabilities of these rows is equal to* 2*, which is the the number of possible combinations of (cardinality of) parent variables* $\mathcal{R}_{2,1}^{pred}$ *and* $\mathcal{E}_2(1,3)$ *(i.e.* $K_{\mathcal{R}_{2,1}^{pred}} \times K_{\mathcal{E}_2(1,3)} = 2 \times 1 = 2$*).*

Table 4.11: CPT of node $\mathcal{C}_{2,1}$

$\mathcal{C}_{2,1}$	$P(\mathcal{C}_{2,1})$
1	0.3
5	0.7

Table 4.12: CPT of node $\mathcal{E}_2(1,3)$

$\mathcal{E}_2(1,3)$	$P(\mathcal{E}_2(1,3))$
1	1

Table 4.13: CPT of node $\mathcal{R}_{2,1}^{pred}$

$\mathcal{C}_{2,1}$	$\mathcal{R}_{2,1}^{pred}$	$P(\mathcal{R}_{2,1}^{pred} \mid \mathcal{C}_{2,1})$
1	1	1
1	5	0
5	1	0
5	5	1

Table 4.14: CPT of node $\mathcal{S}_{2,2}(\tau_{2,1})$

$\mathcal{R}_{2,1}^{pred}$	$\mathcal{E}_2(1,2)$	$\mathcal{S}_{2,2}(\tau_{2,1})$	$P(\mathcal{S}_{2,2}(\tau_{2,1}) \mid \mathcal{R}_{2,1}^{pred}, \mathcal{E}_2(1,2))$
1	0	1	1
1	0	5	0
5	0	1	0
5	0	5	1

Table 4.15: CPT of node $\mathcal{S}_{2,3}(\tau_{2,1})$

$\mathcal{R}_{2,1}^{pred}$	$\mathcal{E}_2(1,3)$	$\mathcal{S}_{2,3}(\tau_{2,1})$	$P(\mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{R}_{2,1}^{pred}, \mathcal{E}_2(1,3))$
1	1	2	1
1	1	6	0
5	1	2	0
5	1	6	1

Table 4.16: CPT of node $\mathcal{M}_{2,2}$

$\mathcal{S}_{2,2}(\tau_{2,1})$	$\mathcal{M}_{2,2}$	$P(\mathcal{M}_{2,2} \mid \mathcal{S}_{2,2}(\tau_{2,1}))$
1	1	1
1	5	0
5	1	0
5	5	1

Table 4.17: CPT of node $\mathcal{M}_{2,3}$

$\mathcal{S}_{2,3}(\tau_{2,1})$	$\mathcal{M}_{2,3}$	$P(\mathcal{M}_{2,3} \mid \mathcal{S}_{2,3}(\tau_{2,1}))$
2	2	1
2	6	0
6	2	0
6	6	1

Table 4.18: CPT of node $\mathcal{R}_{2,2}^{pred}$

$\mathcal{M}_{2,2}$	$\mathcal{C}_{2,2}$	$\mathcal{R}_{2,2}^{pred}$	$P(\mathcal{R}_{2,2}^{pred} \mid \mathcal{M}_{2,2}, \mathcal{C}_{2,2})$
1	3	4	1
1	3	8	0
1	3	12	0
1	7	4	0
1	7	8	1
1	7	12	0
5	3	4	0
5	3	8	1
5	3	12	0
5	7	4	0
5	7	8	0
5	7	12	1

Table 4.19: CPT of node $\mathcal{R}_{2,3}^{pred}$

$\mathcal{M}_{2,3}$	$\mathcal{C}_{2,3}$	$\mathcal{R}_{2,3}^{pred}$	$P(\mathcal{R}_{2,3}^{pred} \mid \mathcal{M}_{2,3}, \mathcal{C}_{2,3})$
2	4	6	1
2	4	10	0
2	4	14	0
2	8	6	0
2	8	10	1
2	8	14	0
6	4	6	0
6	4	10	1
6	4	14	0
6	8	6	0
6	8	10	0
6	8	14	1

Table 4.20: CPT of node $\mathcal{S}_{2,4}(\tau_{2,2})$

$\mathcal{R}_{2,2}^{pred}$	$\mathcal{E}_2(2,4)$	$\mathcal{S}_{2,4}(\tau_{2,2})$	$P(\mathcal{S}_{2,4}(\tau_{2,2}) \mid \mathcal{R}_{2,2}^{pred}, \mathcal{E}_2(2,4))$
4	0	4	1
4	0	8	0
4	0	12	0
8	0	4	0
8	0	8	1
8	0	12	0
12	0	4	0
12	0	8	0
12	0	12	1

Table 4.21: CPT of node $\mathcal{S}_{2,4}(\tau_{2,3})$

$\mathcal{R}_{2,3}^{pred}$	$\mathcal{E}_2(3,4)$	$\mathcal{S}_{2,4}(\tau_{2,3})$	$P(\mathcal{S}_{2,4}(\tau_{2,3}) \mid \mathcal{R}_{2,3}^{pred}, \mathcal{E}_2(3,4))$
6	1	7	1
6	1	11	0
6	1	15	0
10	1	7	0
10	1	11	1
10	1	15	0
14	1	7	0
14	1	11	0
14	1	15	1

Table 4.22: CPT of node $\mathcal{M}_{2,4}$

$\mathcal{S}_{2,4}(\tau_{2,2})$	$\mathcal{S}_{2,4}(\tau_{2,3})$	$\mathcal{M}_{2,4}$	$P(\mathcal{M}_{2,4} \mid \mathcal{S}_{2,4}(\tau_{2,2}),\ \mathcal{S}_{2,4}(\tau_{2,3}))$
4	7	7	1
4	7	8/11/12/15	0
4	11	11	1
4	11	7/8/12/15	0
4	15	15	1
4	15	7/8/11/12	0
8	7	8	1
8	7	7/11/12/15	0
8	11	11	1
8	11	7/8/12/15	0
8	15	15	1
8	15	7/8/11/12	0
12	7	12	1
12	7	7/8/11/15	0
12	11	12	1
12	11	7/8/11/15	0
12	15	15	1
12	15	7/8/11/12	0

Table 4.23: CPT of node $\mathcal{R}_{2,4}^{pred}$

$\mathcal{M}_{2,4}$	$\mathcal{C}_{2,4}$	$\mathcal{R}_{2,4}^{pred}$	$P(\mathcal{R}_{2,4}^{pred} \mid \mathcal{M}_{2,4}, \mathcal{C}_{2,4})$
7	2	9	1
7	2	10/13/14/17	0
8	2	10	1
8	2	9/13/14/17	0
11	2	13	1
11	2	9/10/14/17	0
12	2	14	1
12	2	9/10/13/17	0
15	2	17	1
15	2	9/10/13/14	0

For the sake of clarity, we reduce the size of CPT tables 4.22 and 4.23 by putting multiple possible values of the same variable ($\mathcal{M}_{2,4}$ and $\mathcal{R}_{2,4}^{pred}$ respectively) in the same row since they have the same probability of occurrence that equals 0.

4.3.2 Probabilistic Inference

Probabilistic inference consists in computing the joint probability of some random variables conditioned or not to some other random variables. In this context, the probability distribution that we want to compute is also called a **query**. For instance, to solve the query $P(\mathcal{X}, \mathcal{Y}|\mathcal{Z})$ we calculate the CPT table of the probability distribution $P(\mathcal{X}, \mathcal{Y}|\mathcal{Z})$. In a Bayesian network, inference is considered as a mechanism for applying Bayes' theorem to complex problems with several dependent random variables.

A naive approach for exact probabilistic inference in a Bayesian network is called inference by enumeration. First, it consists in calculating the joint distribution of all random variables involved in the Bayesian graph. This joint distribution is computed by multiplying the distribution of each random variable conditioned to its parents $P(\{\mathcal{A}, \cdots; \mathcal{A} \in V\}) = \prod_{\mathcal{A} \in V} P(\mathcal{A} \mid parents(\mathcal{A}))$. Second, the

query is deduced by summing over non-query variables. This approach results in a large CPT table (factor) of the joint distribution that includes all random variables. This CPT enumerates all combinations of possible values for all variables. Hence, its size is equal to the product of the number of possible values for each random variable, i.e. $\prod_{\mathcal{A} \in V} K_{\mathcal{A}}$.

A more efficient approach for probabilistic inference is known as variable elimination. It tries to avoid creating large CPT tables. In the next Section 4.3.2.1, we use this approach to compute the exact distribution of response times according to Equation 4.4. In addition, there exist some approximation methods for probabilistic inference, such as sampling and Monte Carlo simulation. These methods reduce the computational complexity and manipulate larger Bayesian networks with many more variables. In Section 4.3.2.2, we use sampling to approximate preceding response time distribution.

4.3.2.1 Exact Inference: Variable Elimination

Variable elimination is an exact inference method for Bayesian networks. Instead of joining all variable distributions together as in the inference by enumeration approach, it interleaves joining and elimination (marginalization) of non-query variables. Hence, it avoids creating a large CPT table that represents the joint distribution of all random variables involved in the Bayesian graph. Therefore, variable elimination offers better performance than inference by enumeration approach.

In this section, we use the variable elimination approach to compute the exact distribution of the preceding response time $\mathcal{R}_{i,j}^{pred}$ of a sub-task $\tau_{i,j}$ taking into consideration dependencies between different random variables included in response time Equation 4.4.

Algorithm 2 illustrates how the variable elimination method works. For each hidden (non-query) variable v, we join all factors (CPT tables) that mention that variable v (lines $4-9$). Then, we eliminate this non-query variable v (line 10) and we add the newly obtained factor to the list of factors F (line 11). We repeat these operations (joining and elimination) until all non-query variables have been eliminated (lines $2-12$). After that, we join all the remaining factors to obtain the CPT table of the needed query (called $query\_factor$).

The "join" operation used in Algorithm 2 line 6 and 15 consists in multiplying two factors to obtain the joint distribution according to Bayes' rules $P(\mathcal{X} = x, \mathcal{Y} = y) = P(\mathcal{X} = x) \times P(\mathcal{Y} = y \mid \mathcal{X} = x)$. Indeed, we match rows from the two CPT tables that have the same values of common random variables and we multiply their corresponding probabilities to obtain the probability of the resulting factor.

Algorithm 2: Variable elimination algorithm

Data: list of factor F, list of variable V, query variable Q
Result: distribution of query variable

1 $H = V \setminus \{Q\}$ `/* list of Hidden (non-query) variables */`
2 **foreach** $v \in H$ **do**
3 $\quad new\_factor = \varnothing$ `/* Empty CPT table */`
4 $\quad$ **foreach** $f \in F$ **do**
5 $\quad\quad$ **if** $v \in f$ **then**
6 $\quad\quad\quad new\_factor = \text{join}(new\_factor, f)$
7 $\quad\quad\quad F = F \setminus \{f\}$
8 $\quad\quad$ **end**
9 $\quad$ **end**
10 $\quad new\_factor = \text{eliminate}(new\_factor, v)$
11 $\quad F = F \cup \{new\_factor\}$
12 **end**
13 $query\_factor = \varnothing$ `/* Empty CPT table */`
14 **foreach** $f \in F$ **do**
15 $\quad query\_factor = \text{join}(query\_factor, f)$
16 **end**
17 **return** $query\_factor$

For the "eliminate" operation used in Algorithm 2 line 10 consists in summing over one random variable $\mathcal{Y}$ of a factor that represents a joint distribution $P(\mathcal{X}, \mathcal{Y})$ in order to obtain the marginal distribution according to the following marginalization formula: $P(\mathcal{X}) = \sum_y P(\mathcal{X}, \mathcal{Y} = y)$. Indeed, we group rows, from the CPT table concerned, that have same value of variable $\mathcal{X}$ even if the values of $\mathcal{Y}$ are different and we sum their corresponding probabilities.

Example 4.9. *In this example, we apply the variable elimination algorithm on the Bayesian network defined by the dependency graph in Figure 4.7 and by CPT Tables (from Table 4.13 to Table 4.23). Then, we answer the query* $P(\mathcal{R}_{2,4}^{pred})$ *and we compute the exact distribution of the preceding response time* $\mathcal{R}_{2,4}^{pred}$ *of sub-task* $\tau_{2,4}$.

Eliminating $\mathcal{R}_{2,1}^{pred}$ ***variable:***

First, we join all factors that contain the $\mathcal{R}_{2,1}^{pred}$ *variable. Hence, we join the following factors:* $P(\mathcal{R}_{2,1}^{pred} \mid \mathcal{C}_{2,1})$, $P(\mathcal{S}_{2,2}(\tau_{2,1}) \mid \mathcal{R}_{2,1}^{pred}, \mathcal{E}_2(1,2))$ *and* $P(\mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{R}_{2,1}^{pred}, \mathcal{E}_2(1,3))$. *The result of joining the two factors* $P(\mathcal{R}_{2,1}^{pred} \mid \mathcal{C}_{2,1})$ *and* $P(\mathcal{S}_{2,2}(\tau_{2,1}) \mid \mathcal{R}_{2,1}^{pred}, \mathcal{E}_2(1,2))$ *(CPT Table 4.13 and CPT Table 4.14) is shown in the following Table 4.24.*

Table 4.24: CPT of factor $P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{R}_{2,1}^{pred} \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1})$

$\mathcal{E}_2(1,2)$	$\mathcal{C}_{2,1}$	$\mathcal{S}_{2,2}(\tau_{2,1})$	$\mathcal{R}_{2,1}^{pred}$	$P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{R}_{2,1}^{pred} \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1})$
0	1	1	1	$1 \times 1 = 1$
0	1	1	5	$0 \times 0 = 0$
0	1	5	1	$0 \times 1 = 0$
0	1	5	5	$1 \times 0 = 0$
0	5	1	1	$1 \times 0 = 0$
0	5	1	5	$0 \times 1 = 0$
0	5	5	1	$0 \times 0 = 0$
0	5	5	5	$1 \times 1 = 1$

The result of joining the two factors $P(\mathcal{R}_{2,1}^{pred} \mid \mathcal{C}_{2,1})$ and $P(\mathcal{S}_{2,2}(\tau_{2,1}) \mid \mathcal{R}_{2,1}^{pred}, \mathcal{E}_2(1,2))$ (CPT Table 4.13 and CPT Table 4.14) is presented in the following Table 4.25.

Table 4.25: CPT of factor $P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{R}_{2,1}^{pred}, \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1}, \mathcal{E}_2(1,3))$

$\mathcal{E}_2(1,2)$	$\mathcal{C}_{2,1}$	$\mathcal{E}_2(1,3)$	$\mathcal{S}_{2,2}(\tau_{2,1})$	$\mathcal{R}_{2,1}^{pred}$	$\mathcal{S}_{2,3}(\tau_{2,1})$	$P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{R}_{2,1}^{pred}, \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1}, \mathcal{E}_2(1,3))$
0	1	1	1	1	2	$1 \times 1 = 1$
0	1	1	1	5	2	$0 \times 0 = 0$
0	1	1	5	1	2	$0 \times 1 = 0$
0	1	1	5	5	2	$0 \times 0 = 0$
0	5	1	1	1	2	$0 \times 1 = 0$
0	5	1	1	5	2	$0 \times 0 = 0$
0	5	1	5	1	2	$0 \times 1 = 0$
0	5	1	5	5	2	$1 \times 0 = 0$
0	1	1	1	1	6	$1 \times 0 = 0$
0	1	1	1	5	6	$0 \times 1 = 0$
0	1	1	5	1	6	$0 \times 0 = 0$
0	1	1	5	5	6	$0 \times 1 = 0$
0	5	1	1	1	6	$0 \times 0 = 0$
0	5	1	1	5	6	$0 \times 1 = 0$
0	5	1	5	1	6	$0 \times 0 = 0$
0	5	1	5	5	6	$1 \times 1 = 1$

Second, we sum over $\mathcal{R}_{2,1}^{pred}$ variable in order to eliminate $\mathcal{R}_{2,1}^{pred}$ from the join factor $P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{R}_{2,1}^{pred}, \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1}, \mathcal{E}_2(1,3))$ (CPT Table 4.25). Hence, we obtain the new factor $P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1}, \mathcal{E}_2(1,3))$ given in the following Table 4.26.

Table 4.26: CPT of factor $P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1}, \mathcal{E}_2(1,3))$

$\mathcal{E}_2(1,2)$	$\mathcal{C}_{2,1}$	$\mathcal{E}_2(1,3)$	$\mathcal{S}_{2,2}(\tau_{2,1})$	$\mathcal{S}_{2,3}(\tau_{2,1})$	$P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1}, \mathcal{E}_2(1,3))$
0	1	1	1	2	$1+0=1$
0	1	1	5	2	$0+0=0$
0	5	1	1	2	$0+0=0$
0	5	1	5	2	$0+0=0$
0	1	1	1	6	$0+0=0$
0	1	1	5	6	$0+0=0$
0	5	1	1	6	$0+0=0$
0	5	1	5	6	$0+1=1$

Eliminating $\mathcal{C}_{2,1}$ variable:

The result of joining the two factors $P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{C}_{2,1}, \mathcal{E}_2(1,3))$ and $P(\mathcal{C}_{2,1})$ (CPT Table 4.26 and Table 4.11 respectively) that contain $\mathcal{C}_{2,1}$ variable is presented in the following Table 4.27.

Table 4.27: CPT of factor $P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{S}_{2,3}(\tau_{2,1}), \mathcal{C}_{2,1} \mid \mathcal{E}_2(1,2), \mathcal{E}_2(1,3))$

$\mathcal{E}_2(1,2)$	$\mathcal{E}_2(1,3)$	$\mathcal{S}_{2,2}(\tau_{2,1})$	$\mathcal{S}_{2,3}(\tau_{2,1})$	$\mathcal{C}_{2,1}$	$P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{S}_{2,3}(\tau_{2,1}), \mathcal{C}_{2,1} \mid \mathcal{E}_2(1,2), \mathcal{E}_2(1,3))$
0	1	1	2	1	$1 \times 0.3 = 0.3$
0	1	1	2	5	$0 \times 0.7 = 0$
0	1	5	2	1	$0 \times 0.3 = 0$
0	1	5	2	5	$0 \times 0.7 = 0$
0	1	1	6	1	$0 \times 0.3 = 0$
0	1	1	6	5	$0 \times 0.7 = 0$
0	1	5	6	1	$0 \times 0.3 = 0$
0	1	5	6	5	$1 \times 0.7 = 0.7$

After summing over $\mathcal{C}_{2,1}$ variable, we obtain a new factor given in the following Table 4.28.

Table 4.28: CPT of factor $P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{E}_2(1,3))$

$\mathcal{E}_2(1,2)$	$\mathcal{E}_2(1,3)$	$\mathcal{S}_{2,2}(\tau_{2,1})$	$\mathcal{S}_{2,3}(\tau_{2,1})$	$P(\mathcal{S}_{2,2}(\tau_{2,1}), \mathcal{S}_{2,3}(\tau_{2,1}) \mid \mathcal{E}_2(1,2), \mathcal{E}_2(1,3))$
0	1	1	2	$0.3+0=0.3$
0	1	5	2	$0+0=0$
0	1	1	6	$0+0=0$
0	1	5	6	$0+0.7=0.7$

Eliminating remaining non-query variables:

After eliminating all non-query variables, we obtain the exact distribution of the preceding response time $\mathcal{R}_{2,4}^{pred}$ of sub-task $\tau_{2,4}$ (CPT Table 4.29). Since $\tau_{2,4}$ does not have parallel sub-tasks (i.e. $\Pi_{2,4}^{pred} = \varnothing$), then $\mathcal{R}_{2,4}^{isol} = \mathcal{R}_{2,4}^{pred}$ and the distribution of the response time in isolation is also given by the CPT Table 4.29. We note that this distribution computed based on a Bayesian network is equal to the exact one obtained by exploring all combinations of possible values of pWCETs and pWCCTs given in Table 4.4 (on page 71).

Table 4.29: CPT of factor $P(\mathcal{R}_{2,4}^{pred})$

$\mathcal{R}_{2,4}^{pred}$	$P(\mathcal{R}_{2,4}^{pred})$
9	0.018
10	0.162
13	0.162
14	0.378
17	0.28

The efficiency of the variable elimination algorithm depends on the elimination order of variables because some orders may cause large factors (CPT tables) than other orders. Hence, we should use the elimination order that reduces the size of created CPT tables in order to guarantee better performance. However, finding the best elimination order is known to be an NP-hard problem [89]. In addition, Cooper [90] proves that the 3-SAT problem could be reduced to the exact inference in Bayesian networks. Therefore, exact inference is also an NP-hard problem.

We deduce that even if the best elimination order is found, then the run-time of the variable elimination algorithm could be exponential in the size of the Bayesian network for some cases and some graph structures. Despite the NP-hardness, there exist some graph structures like the polytree where the variable elimination algorithm runs in time linear in the size of the network [91].

4.3.2.2 Approximate Inference: Sampling

Since exact inference is an NP-hard problem [90], we resort to approximate inference and we use sampling techniques in order to approximate the distribution of a random variable (or a query) in a Bayesian network. Indeed, sampling helps to process larger Bayesian networks than exact inference does because its complexity is linear in the number of variables.

To compute a response time distribution using a Bayesian network, we use a simple query without conditional variables. This query is composed of a single variable (i.e. the needed response time). Hence, we use forward sampling [92] because it works well when no conditional variables are included in the query (i.e. no evidence is observed) [93]. Forward sampling samples from the joint distribution of all random variables. Then, it estimates the probability distribution of the query variable by counting the frequencies of each possible value from the samples obtained.

To create a sample, we sample variables in topological order. We start by sampling the variables with no parents, then we move to their successors and we sample from their CPT tables conditioned to parents' values already sampled at the previous step. We proceed like this until all random variables have been sampled. In addition, we use inverse transform sampling to sample from a distribution or a CPT table; we sample from uniform distribution and we use the inverse CDF function to generate a sample from the given distribution.

Example 4.10. *Table 4.30 illustrates some samples of preceding response times of sub-tasks belonging to task τ_2 in Figure 4.7 (on page 83). These samples are obtained with the forward sampling method.*

Table 4.30: Example of samples generated with the forward sampling method

Sample	$\mathcal{R}_{2,1}^{pred}$	$\mathcal{R}_{2,2}^{pred}$	$\mathcal{R}_{2,3}^{pred}$	$\mathcal{R}_{2,4}^{pred}$
1	5	12	11	14
2	5	12	15	17
3	5	12	11	14
4	1	8	11	13
5	1	8	7	10
6	5	12	11	14
7	5	12	11	14
8	1	8	7	10
9	1	4	7	9
10	5	8	11	13
		⋮		
N_s	5	12	11	14

To estimate the probability of each value for a given random variable, we divide the number of occurrences by the total number of samples N_s. Hence, we obtain different approximations of preceding response time distributions using different numbers of samples N_s. These approximations are given in Table 4.31.

Table 4.31: Preceding response time distributions for sub-tasks of task τ_2 estimated from samples.

Sub-task	$N_s = 10$	$N_s = 100$	$N_s = 1000$	$N_s = 10000$
$\tau_{\mathbf{2,1}}$	$\begin{pmatrix} 1 & 5 \\ .4 & .6 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 \\ .32 & .68 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 \\ .311 & .689 \end{pmatrix}$	$\begin{pmatrix} 1 & 5 \\ .2994 & .7006 \end{pmatrix}$
$\tau_{\mathbf{2,2}}$	$\begin{pmatrix} 4 & 8 & 12 \\ .1 & .4 & .5 \end{pmatrix}$	$\begin{pmatrix} 4 & 8 & 12 \\ .04 & .42 & .54 \end{pmatrix}$	$\begin{pmatrix} 4 & 8 & 12 \\ .029 & .330 & .641 \end{pmatrix}$	$\begin{pmatrix} 4 & 8 & 12 \\ .0304 & .3429 & .6267 \end{pmatrix}$
$\tau_{\mathbf{2,3}}$	$\begin{pmatrix} 6 & 10 & 14 \\ .3 & .6 & .1 \end{pmatrix}$	$\begin{pmatrix} 6 & 10 & 14 \\ .24 & .52 & .24 \end{pmatrix}$	$\begin{pmatrix} 6 & 10 & 14 \\ .173 & .542 & .285 \end{pmatrix}$	$\begin{pmatrix} 6 & 10 & 14 \\ .1821 & .5393 & .2786 \end{pmatrix}$
$\tau_{\mathbf{2,4}}$	$\begin{pmatrix} 9 & 10 & 13 & 14 & 17 \\ .1 & .2 & .2 & .4 & .1 \end{pmatrix}$	$\begin{pmatrix} 9 & 10 & 13 & 14 & 17 \\ .02 & .2 & .21 & .35 & .22 \end{pmatrix}$	$\begin{pmatrix} 9 & 10 & 13 & 14 & 17 \\ .018 & .162 & .178 & .364 & .278 \end{pmatrix}$	$\begin{pmatrix} 9 & 10 & 13 & 14 & 17 \\ .018 & .1587 & .163 & .3814 & .2789 \end{pmatrix}$

In Table 4.32, we measure the run-time of the sampling algorithm using different numbers of samples N_s. We note that the run-time increases if we use a bigger number of samples for the approximation.

Table 4.32: Run-time of sampling algorithm using different numbers of samples N_s

$\mathbf{N_s}$	10	100	1000	10000
Run-time (second)	0.024	0.039	0.22	2.157

From the previous example, we deduce that the precision of the distribution approximation depends on the number of samples used, N_s. For instance, with $N_s = 1000$ samples, the highest precision that we could get is three digits after the decimal point. Hence, increasing the number of samples helps to enhance precision but it also increases the run-time of the sampling algorithm because its complexity is also linear with the number of samples.

Even if we increase the precision of the computed distribution, the sampling algorithm still provides an approximation that could under-estimate or over-estimate the exact distribution. Thus, the computed distribution of the response time using sampling approach is not guaranteed to be a safe approximation because it could under-estimate the exact one.

4.4 Schedulability in Probabilistic C-space

In the previous Section 4.3, we study the schedulability and we compute the response time distribution of a DAG task using Bayesian network inference. This approach consists in applying some operations (e.g. convolution, join, eliminate) directly on the timing parameter distributions (pWCETs, pWCCTs). It evaluates the exact distribution of response time and the exact schedulability probability (or

the Deadline Miss Probability) of each DAG task. However, it requires a high computational complexity due to the NP-hardness of the exact inference problem.

In this section, we study the schedulability of probabilistic DAG tasks using deterministic schedulability condition (i.e. $R_i^{global} \leq D_i$). In fact, we apply the deterministic schedulability test on several deterministic task sets with timing parameters equal to different combinations of possible values for timing parameters. Then, we deduce the schedulability probability of the studied DAG task.

Remark. *The schedulability condition could be verified only on one DAG task to study its schedulability. Alternatively, it could be applied on all DAG tasks to study the schedulability of the whole task set.*

4.4.1 C-space and schedulability

In order to visualize the schedulability condition as a region, we use a multi-dimensional space called C-space [83, 84]. This space represents on each dimension different possible values of a timing parameter distribution (e.g. pWCET and pWCCT). Each point in C-space represents a task set with timing parameters equal to the values of this point on each dimension. This task set could be schedulable or non-schedulable. Hence, C-space is divided into two disjoint regions: schedulable and non-schedulable. More formally these regions can be defined as follows.

Definition 4.14 ([83])**.** *For a task set τ that has p timing parameters with several possible values for each of them, the schedulability region in the C-space of p dimensions is the set of p-tuples (of points) such that the defined task set, with timing parameters equal to one of these tuples, is schedulable.*

Similarly, we define the non-schedulable region as the set of p-tuples such that the corresponding task set is not schedulable.

Figure 4.8 shows an example of a C-space with two dimensions. The x axis represents possible values taken by execution time $C_{1,1}$ while the y axis represents values of $C_{1,2}$. Green points correspond to timing parameters that define schedulable task sets and red points refer to non-schedulable task sets.

In the literature, C-space is mainly used in the context of sensitivity analysis [83]. It allows us to visualize the effect of varying some timing parameters. C-space representation also helps to evaluate how much execution time of a given sub-task should be decreased to reach the schedulability region or how much it could be increased while remaining schedulable. These quantities help the system designer

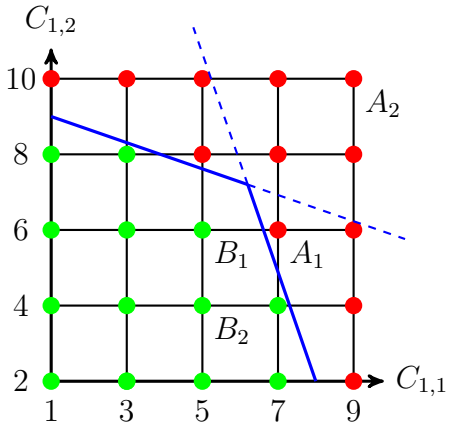


Figure 4.8: Example of a C-space

to determine necessary correction actions to make a system schedulable or possible product extension while guaranteeing schedulability.

In this section, we use the probabilistic version of C-space [94] where each point (i.e. combination of timing parameters) is characterized by its probability of occurrence. This probability is given by the joint distribution of all probabilistic timing parameters. In the case of independent parameters' distributions, the joint distribution is equal to the product of all distributions. Otherwise, the joint distribution is equal to the product of each parameter's probability distribution given its parents, similarly to inference by enumeration on a Bayesian network that captures dependencies between probabilistic timing parameters. In both cases, we note that computing the probability of a point in C-space is linear with respect to the number of dimensions p since this probability is a product of p terms.

For the sake of clarity, we reduce the number of dimensions of C-space by reducing the number of probabilistic timing parameters (several possible values). For this purpose, we assume that communication times between sub-tasks are deterministic parameters (a single possible value). Hence, the C-space has only one dimension per sub-task that represents the pWCET of that sub-task. However, a task set could have more probabilistic parameters than only pWCETs. In this case, we just need to add new dimensions to C-space to handle these new probabilistic parameters.

The schedulability probability of a DAG task is equal to the sum of the probabilities of all points belonging to the schedulable region. The number of these points may be very large because it increases exponentially to the number of dimensions in C-space. Thus, computing the exact sum of all points in schedulable region may be not feasible. In such a case, we resort to sampling techniques. Indeed, we sample random points from C-space and we compute the sum of the probabilities

of schedulable points among sampled points. Then, we normalize by the total sum of the probabilities of all sampled points in order to estimate the schedulability probability. This approach does not guarantee a safe approximation and it may cause an over-estimation of the exact probability of schedulability. Avoiding possible over-estimation of schedulability probability will be the subject of future work.

As mentioned, the schedulability probability is equal to the sum of the probabilities of all schedulable points, so it is sufficient to delimit the schedulable region in order to compute the schedulability probability. Determining the schedulable region and the border between the two regions could be seen as a binary classification problem in C-space where each point is labeled as schedulable or non-schedulable.

4.4.2 C-space and Classification

In this part, we use some properties of C-space and schedulability test to study the type of the border between the two classes (schedulable and non-schedulable regions). Then, we apply an SVM classifier that is appropriate for such a classification problem in order to determine the border.

4.4.2.1 Border and regions characterization

In Section 3.2.2 (on page 33), we point out that RTA is sustainable with respect to the period. From the work of Baruah and Burns [19], we deduce that our RTA is also sustainable with respect to the execution time. We state that our schedulability test is **C-sustainable**. Indeed, if a task set with given timing parameters is schedulable, then, by decreasing the execution time of any, some or all sub-tasks, the task set remains schedulable. Conversely, if it is not schedulable, then, by increasing the execution time of any, some or all sub-tasks, the task set remains non-schedulable.

Graphically, in Figure 4.8, the point A_2 with coordinates $(9, 10)$ should be non-schedulable (red) since it has higher execution times than a non-schedulable point A_1 with coordinates $(7, 6)$. Moreover, the point B_2 with coordinates $(5, 4)$ should be schedulable (green) because it has a lower execution time than its neighbor point B_1 with coordinates $(5, 6)$ that is schedulable. A neighbor point is defined as follows.

Definition 4.15. *In a C-space with p dimensions, we call a neighbor point to a given point A, any point that has its value on one dimension shifted by one step (higher or lower) with regard to point A and that has the same values as point A on the other $p - 1$ dimensions.*

The C-sustainability of schedulability test allows us to prove some properties of schedulable and non-schedulable regions as follows.

Theorem 4.4. *If a schedulability test is C-sustainable, then the corresponding schedulable region (respectively non-schedulable region) in C-space is connected [95].*

Proof. In order to prove that the schedulable region is connected, we prove that any two schedulable points are connected by moving through successive neighbors that are all schedulable (path-connected [96]).

Let A and B be two schedulable points in C-space. We define the point $M = min(A, B)$ such that its value on each dimension is equal to the minimum between values of points A and B on that dimension. From point A, we can reach the point M by moving successively to the neighbor that reduces the value by one step on one of the dimensions with a different value compared to point M. All these neighbors are schedulable due to the C-sustainability property.

Likewise, by moving from a schedulable neighbor to another schedulable neighbor, we can reach point M from point B. Hence, we deduce that we can reach point B from point A. First, we move successively through schedulable neighbors toward point M. Second, from point M, we move to B using the reverse path from B to M composed of schedulable neighbors.

More formally, we could construct a continuous function f from interval $[0, 1]$ to C-space such that $f(0) = A$ and $f(1) = B$. This function maps the interval $[0, 0.5]$ to the line segment $\overline{AM}$. All points of this segment have lower execution times than schedulable point A. Due to the C-sustainability property, they are all schedulable points. Hence, segment $\overline{AM}$ belongs to the scheduled region. Moreover, function f maps the interval $[0.5, 1]$ to the line segment $\overline{MB}$ that also belongs to the scheduled region. We note that function f is continuous and $f(0.5) = M$.

We deduce that the schedulable region is path-connected and consequently it is connected. Similarly, we can prove that the non-schedulable region is also connected. ■

From Theorem 4.4, we deduce that the schedulable and non-schedulable regions in C-space are both connected regions because the used schedulability test based on RTA is C-sustainable. Since these two regions are disjoint, then we could find a single straight line (hyperplane) or a curved line (hypersurface) that separates the two regions.

4.4.2.2 SVM classifier

In machine learning, SVM is a binary classification technique [86]. It constructs a hyperplane that separates data points into two classes. This hyperplane should maximize the margin between the two classes. Thus, SVM classifier is also known as a maximum-margin classifier. However, in some cases, data points are not separable by a linear hyperplane. To resolve this problem, we map the original space into a much higher-dimensional space using a kernel function [85]. In this new space, a separating hyperplane could exist and we could apply the maximum-margin classifier. We note that if the separating hyperplane exists in the original space then, we just use a linear kernel (identity function).

We deduce that it is reasonable to use SVM classifier to determine the border between schedulable and non-schedulable regions in C-space because these two regions could be separated by a single line.

In addition, SVM classifier determines a border between two regions based on the nearest points to that border. These points are called "Support Vectors". However, the number and the positions of far points do not affect the found border. Hence, we just need to study the schedulability and to label only some points that lie near to the border rather than exploring all points in the C-space. This allows us to reduce the computational complexity for determining the schedulable region and for calculating the schedulability probability.

Single core processor

In this part, we study the border between schedulable and non-schedulable regions in C-space when the corresponding task set executes on a single core processor.

Theorem 4.5. *Let a task set τ be executed on a single core. If we use a schedulability test based on RTA then there is a hyperplane that separates the schedulable and non-schedulable regions in C-space.*

Proof. In the preceding response time Equation 3.21 (on page 48) of sub-task $\tau_{i,j}$, the maximum term represents the time required by all predecessors of $\tau_{i,j}$ to finish their execution. In the case of a single core processor, there is no possible parallel execution and all predecessors are executed on the same core. Thus, this maximum term is equal to the sum of the execution times of all predecessors.

On the other hand, the response time in isolation and the global response time of sub-task $\tau_{i,j}$ (Equations 3.22 and 3.23 respectively) are obtained by adding, to the preceding response time, the sum of the execution times of parallel or higher priority sub-tasks. Hence, we deduce that the resulting global response time of a

DAG task is equal to the sum of the execution times of several sub-tasks. In other words, the global response time of task τ_i could be written as a linear combination of some timing parameters i.e. linear combination of execution times of some sub-tasks because communication delays are assumed to be deterministic (constant); $R_i^{global} = \sum_{p,q} a_{p,q} \cdot C_{p,q} + B$. where $a_{p,q} \in \{0, 1, 2, \cdots\}$ represents the number of times sub-task $\tau_{p,q}$ is executed between the release time and the end of the execution of DAG task τ_i. B is a constant term that represents the communication times between different sub-tasks of τ_i.

The schedulability test based on RTA consists in verifying whether the response time of DAG task τ_i is less than or equal to its deadline i.e. $R_i^{global} \leq D_i$. If we replace R_i^{global} by its linear combination formulation, we obtain $\sum_{p,q} a_{p,q} \cdot C_{p,q} + B \leq D_i$. This equation defines the schedulable region in C-space. This region is delimited by the hyperplane defined by the following equation $\sum_{p,q} a_{p,q} \cdot C_{p,q} = D_i - B$. Thus, we find a hyperplane that divides the C-space into schedulable and non-schedulable regions.

■

Example 4.11. *Let τ_1 and τ_2 be two DAG tasks where each is composed of a single sub-task. Their respective periods and deadlines are $T_1 = D_1 = 10$ and $T_2 = D_2 = 12$. We assume that τ_1 has higher priority than τ_2. The sub-tasks $\tau_{1,1}$ and $\tau_{2,1}$, belonging respectively to DAG tasks τ_1 and τ_2, are executed on a single core processor.*

Figure 4.9 represents the C-space corresponding to the task set described above. This C-space has two dimensions that represent the execution times $C_{1,1}$ and $C_{2,1}$ of sub-tasks $\tau_{1,1}$ and $\tau_{2,1}$ respectively.

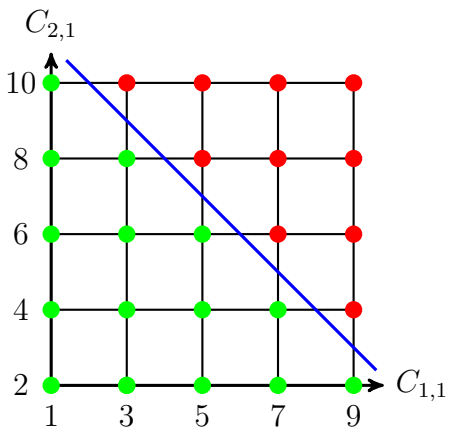


Figure 4.9: C-space of a task set executed on a single core

The response time of τ_2 is equal to the sum of execution times $C_{1,1}$ and $C_{2,1}$ because τ_1 has higher priority and could preempt τ_2 i.e. $R_2^{global} = C_{1,1} + C_{2,1}$. The schedulability test of task τ_2 consists of comparing its response time to its deadline. Hence, we obtain this equation $C_{1,1} + C_{2,1} \leq 12$ that characterizes schedulable (green) points in C-space. The equation of the border between schedulable and non-schedulable regions is given by $C_{1,1} + C_{2,1} = 12$ and it defines the blue line (hyperplane) in Figure 4.9

From the Theorem 4.5, we deduce that we could use a linear kernel for the SVM classifier, in the case of a task set executed on a single core processor, in order to find the exact border (hyperplane) between schedulable and non-schedulable regions in C-space when using a schedulability test based on RTA.

Multi-core processor

In the case of a multi-core processor, parallel sub-tasks could be executed concurrently if they are mapped to different cores. Hence, the maximum term in preceding response time Equation 3.21 (on page 48) is applied between different linear combinations. Each of these linear combinations includes the execution times of some predecessor sub-tasks according to the structure of the graph and the partitioning. Depending on the execution time value of different sub-tasks, the maximum could be caused by one of these linear combinations. In other words, in each sector of C-space, schedulable points are delimited with a different hyperplane. For instance, in Figure 4.8, schedulable points are delimited with a given line (hyperplane) when $C_{1,1} \leq 6$ and they are delimited by another line for $C_{1,1} \geq 6$.

Conversely to the case of single core processors, we deduce that using SVM classifier with a linear kernel does not guarantee that we will find the exact border between schedulable and non-schedulable regions. Therefore, we use a Gaussian kernel in the case of multi-core processors because such a kernel helps to fit complex borders.

From Figure 4.10, we note that some points could be classified wrongly. For instance, if we use the dashed blue line as the border between schedulable and non-schedulable regions, then the red point with coordinates $(9, 2)$ is classified as schedulable while actually it is not schedulable. Even if we use a Gaussian kernel, some points may be classified wrongly, especially in a high dimensional C-space.

In order to avoid classifying non-schedulable points as schedulable, we shift the border toward the schedulable region like in Figure 4.10. Hence, points near to the border that are situated in the schedulable side with respect to the previous border (dashed line), will be classified as non-schedulable with the shifted border (solid

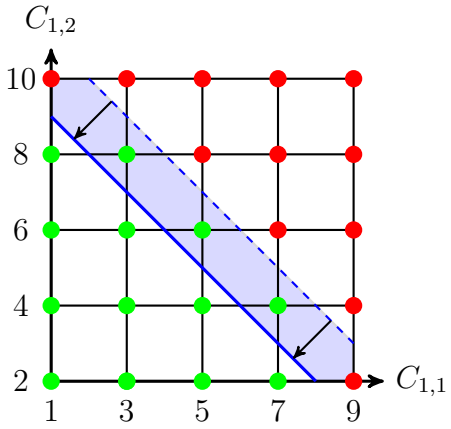


Figure 4.10: Shifting border in C-space to reduce non-schedulable points classified wrongly

line). This approach allows us to reduce the number of non-schedulable points that are labeled as schedulable, which enhance the safety of the schedulability analysis and avoid an over-estimation of the schedulability probability. On the other hand, this approach increases the pessimism by classifying schedulable points as non-schedulable. Thus, it may cause an under-estimation of the schedulability probability, which presents a trade-off between safety and pessimism. In Chapter 6, we study the performance of this approach and the existing trade-off using a confusion matrix [97].

4.5 Conclusion

In this Chapter, we studied the schedulability of a DAG task model with probabilistic execution times and communication delays. We proposed several methods to estimate the response time distribution and the schedulability probability.

First, we defined probabilistic maximum operators and we extended the deterministic response time equations proposed in Chapter 3 to deal with discrete probability distributions that represent timing parameters. The probabilistic maximum operators used are either very pessimistic or require independence between different random variables involved in the response time equations. This independence is only guaranteed for some specific structure of precedence graph (e.g. arborescence and polytree). Therefore, in the second section, we used a Bayesian network to model the dependencies between the different random variables, which allowed us to compute the exact response time distribution. Moreover, we studied the schedulability of a probabilistic DAG task model by representing probabilistic timing parameters and schedulability condition as regions in the C-space. Then, we used deterministic schedulability test from Chapter 3 and SVM classification

techniques to determine the schedulable region in C-space and to estimate the schedulability probability.

From the response time distribution and schedulability probability provided by our probabilistic schedulability analyses, we compute the DMP of the real-time system studied. This DMP is proportional to the failure rate of the system. Thus, we compare this rate to the threshold required to ensure safety and to validate the system. For instance, let a real-time system be integrated in a safety-critical functionality of a vehicle such as braking. This functionality belongs to the highest Automotive Safety and Integrity Level (ASIL D) that should guarantee a failure rate per hour less than 10^{-8} according to the standard ISO-26262 [10]. If the failure rate corresponding to the DMP obtained by the schedulability analysis, is less than the required threshold 10^{-8}, then we could deem the system as feasible and safe regarding to the timing behavior. Otherwise, the system does not reach the required level of safety (ASIL D). Hence, it is not feasible and not validated.

5 Scheduling techniques

Contents

In this chapter, we propose scheduling techniques for the DAG task model following a partitioned and fixed-priority policy. First, we study the problem of priority assignment in Section 5.1. To do so, we distinguish between defining priority at the task level and at the sub-task level. We adapt existing priority assignment policies for independent tasks to the DAG task model. At the sub-task level, existing priority assignment heuristics are applied on a non partitioned DAG. Hence, we propose new priority assignment algorithms that take into account the sub-task partitioning. Second, we tackle the partitioning problem in Section 5.2. Indeed, we propose a partitioning heuristic that operates on multiple DAG tasks with different periods. This heuristic assigns each sub-task to a given core while balancing the load between cores in such a way to maximize possible parallelism. Third, in Section 5.3, we present the idea of reducing the size of a DAG by merging some sub-tasks without modifying the precedence constraints. This reduction allows us to decrease the computational complexities of other scheduling and schedulability techniques

since all of them depend on the number of sub-tasks. Finally, in Section 5.4, we describe a scheduling workflow that joins different scheduling techniques proposed in order to reduce the response time and enhance the schedulability of DAG tasks.

Remark. *The algorithms proposed in this chapter operate on deterministic as well as probabilistic task models. In the case of probabilistic parameters, we use expected values of the probability distributions instead of the deterministic values.*

5.1 Priority Assignment

Scheduling algorithms allocate shared resources (like CPU, communication bus, disk drive, etc.) to competing tasks. They may use best-effort policies based on time-sharing and fairness like round-robin [98] and fair queuing [99] scheduling. These scheduling approaches improve efficiency by minimizing resource starvation and prevent a task from waiting for the resource infinitely. For instance, round-robin policy executes each job for a time slice (called also quantum) then it moves to the next job in a circular queue. It repeats this until all the jobs are finished. Round-robin prevents starvation and shares resources fairly among tasks. However, response and waiting times of a given task may be relatively large because they mainly depend on time slices and the number of tasks in the whole system and not on individual characteristics of a task.

Conversely, in real-time scheduling, some tasks have high rates of activation and should be more reactive with a small waiting and response time. Hence, we use priority-driven scheduling that allows us to reduce the response time for crucial and demanding tasks by assigning high priority to them. We also focus on fixed-priority policies to reduce interactions between different tasks and cores compared to dynamic priority assignment. Reducing these interactions allows us to decrease over-estimation and pessimism in the response time analysis.

When applying priority-driven scheduling algorithms on a DAG task model, priority could be defined only at the task level or at both the task and sub-task levels. If a DAG task τ_i has higher priority than a DAG task τ_p, then all sub-tasks of τ_i have higher priorities than all sub-tasks of τ_p. In the case of priority defined at the task level only, all sub-tasks of DAG task τ_i have the same priority as τ_i. To define priorities of DAG tasks, we could use one of the priority assignment algorithms from the literature, such as *Rate Monotonic* [12], *Deadline Monotonic* [13] and *Audsley's algorithm* [75, 76].

On the other hand, if priorities are also defined at the sub-task level then, each sub-task inside the same graph has an individual priority that is different from

other sub-tasks. These individual priorities define a kind of execution order between parallel sub-tasks. Depending on the resulting execution order, the response time of the DAG task is affected, and it could be reduced or increased.

In this section, we assume that sub-tasks to cores mapping has been already established and is given. In the first part, we consider fixed-priority assignment policies that we use to assign priorities at the task level. In the second part, we tackle priority assignment at the sub-task level for each DAG task. This problem consists of finding the best execution order of sub-tasks inside the same DAG to reduce the response time of the DAG task considered.

5.1.1 Priority assignment at the task level

In this section, we define a priority for each DAG task using two priority assignment policies. First, we assign a priority for a DAG task based only on its individual characteristics and we apply the *Deadline Monotonic* policy that prioritizes the DAG task with the lowest relative deadline. Second, we determine the priority of a DAG task depending on the parameters and structures of all DAG tasks. Indeed, we apply *Audsley's algorithm* using proposed response time analysis as a schedulability test to compare different priority orderings.

5.1.1.1 Deadline Monotonic

Deadline Monotonic is a fixed-priority scheduling policy. It orders and prioritizes tasks in the increasing order of their relative deadline. This algorithm gives higher priority for the task with a lower relative deadline in order to reduce its response time. Hence, the task could finish its execution before its deadline and respect its temporal constraints. The computational complexity of this approach is equivalent to a sorting problem (i.e. $O(n \times \log n)$). In the task model studied, we consider a constrained deadline ($D_i \leq T_i$). In the case where the deadline is equal to the period ($D_i = T_i$), the deadline monotonic priority assignment policy becomes equivalent to Rate Monotonic [12].

Leung and Whitehead [13] show that deadline monotonic is an optimal policy for fixed-priority scheduling on a uniprocessor system with sporadic arrivals and constrained deadlines. Since partitioned multiprocessor scheduling could be seen as several uniprocessor scheduling problems [14], we decide to use the deadline monotonic algorithm in order to benefit from its good performance reported in the literature. However, Deadline Monotonic does not guarantee optimality on partitioned multiprocessor because the partitioning problem is similar to the bin

packing problem, which is NP-hard [60]. Even if Deadline Monotonic is optimal for single core processors, existing heuristics-based solutions for the partitioning problem do not guarantee optimality.

Example 5.1. *In this example, we illustrate how Deadline Monotonic priority assignment could enhance reactivity of demanding tasks and allow them to finish earlier and respect their deadlines.*

Table 5.1: Parameters of the task set described in Figure 5.1

Sub-task	$C_{i,j}$	D_i	T_i	Precedence	delay
$\tau_{1,1}$	9	11	30	$e_1(1,2)$	1
$\tau_{1,2}$	1	11	30	$e_2(1,2)$	0
$\tau_{2,1}$	1	15	30	$e_2(1,3)$	1
$\tau_{2,2}$	1	15	30	$e_2(2,4)$	2
$\tau_{2,3}$	2	15	30	$e_2(3,4)$	0
$\tau_{2,4}$	2	15	30	–	–

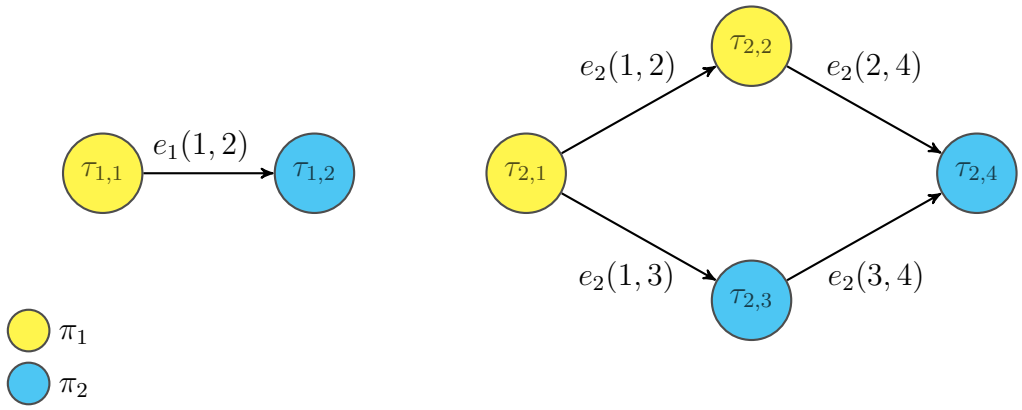
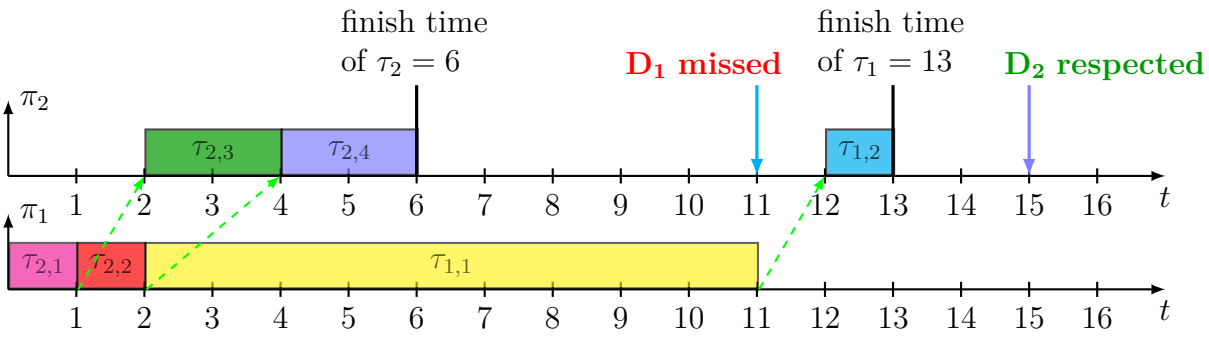


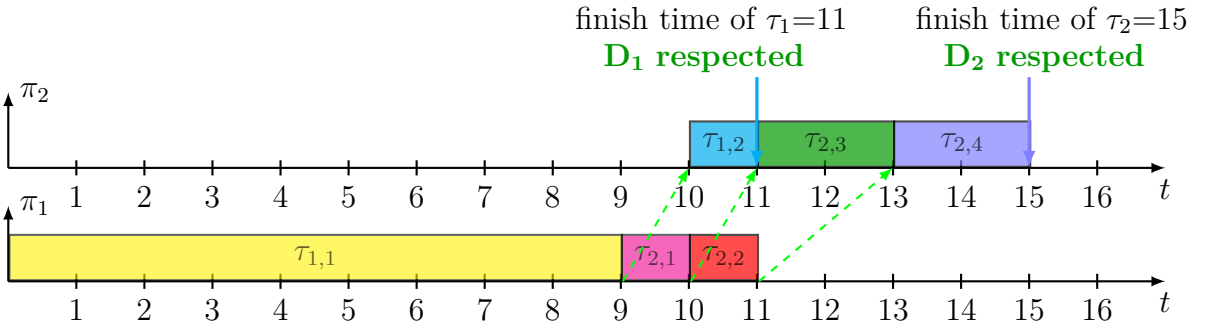
Figure 5.1: Example of a DAG task set with sub-tasks assigned to different cores

Figure 5.1 and Table 5.1 represent an example of a task set composed of two DAG tasks scheduled on two cores with a preemptive and fixed-priority policy. In Figure 5.2, we illustrate two different schedules for the previous task set with two possible priority assignment strategies at the task level. We note that green dashed arrows represent communication delays between sub-tasks. In the first schedule (see Figure 5.2a), we assume that τ_2 has higher priority than τ_1. We note that DAG task τ_2 finishes its execution at time instant $t = 6$ before its deadline $D_2 = 15$. While, τ_1 finishes its execution at $t = 13$ and misses its deadline $D_1 = 11$ because it is delayed by sub-tasks $\tau_{2,1}$ and $\tau_{2,2}$ on core π_1. In the second schedule (see Figure 5.2b), priorities are assigned according to the DM policy where τ_1 has higher priority than

τ_2 *because its deadline is lower* $D_1 < D_2$*. In this case, we note that both DAG tasks respect their deadline. Indeed,* τ_1 *finishes its execution at* $t = 11$ *and* τ_2 *finishes its execution at* $t = 15$*.*



(a) First scheduling: τ_2 has higher priority than τ_1



(b) Second scheduling: τ_1 has higher priority than τ_2 (according to DM policy)

Figure 5.2: Scheduling of task set defined by Figure 5.1 and Table 5.1 with two different priority assignments

5.1.1.2 Audsley's Algorithm

Audsley's algorithm [75, 76] is a priority assignment policy for tasks. This algorithm was originally devised for fixed-priority and preemptive scheduling on a single core (processor). Audsley's algorithm uses a given schedulability test to derive a priority assignment that guarantees the schedulability of all tasks according to the test used. The idea of this algorithm is based on the following Theorem 5.1:

Theorem 5.1 ([76])**.** *Let* $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *be a set of* n *periodic tasks scheduled with fixed-priority and preemptive policy on a single core processor. Each task has an individual priority level* $\in \{1, 2, \ldots, n\}$ *where the* n^{th} *level is the lowest one. We assume that the tasks assigned to priority levels from* $i, \ldots, n$ *are feasible under these priority levels while the other tasks are not assigned to any priority level.*

Then, a feasible priority ordering exists for all tasks, if and only if a feasible priority ordering exists that assigns the same tasks to priority levels $i, \ldots, n$.

By applying Theorem 5.1, we could build a feasible priority ordering. First, we look for the task that is feasible at the lowest priority level n. According to Theorem 5.1, if a feasible priority ordering exists, then we should find a feasible ordering that assigns that task to the priority level n. Second, we seek a task, from the remaining tasks, that is feasible at the priority level $n-1$ and we assign this task to level $n-1$. Third, we proceed as previously until all tasks have been assigned or until a feasible assignment cannot be found. The approach described above is a greedy algorithm that assigns, for each priority level, one of the feasible tasks at that level. This approach is implemented through the following Algorithm 3.

Algorithm 3: Priority assignment at the task level with Audsley's algorithm

Data: Task set τ, n number of task, m number of core and $\pi(.)$ mapping
Result: Task set schedulability and priority order

```
1  priority = zeros(n)
2  τ' = τ
3  for l ∈ {n...1} do
4  |  assigned = False
5  |  for τ_i ∈ τ' do
6  |  |  if feasible(τ_i, l) then
7  |  |  |  priority[i] = l
8  |  |  |  τ' = τ' \ {τ_i}
9  |  |  |  assigned = True
10 |  |  |  break
11 |  |  end
12 |  end
13 |  if assigned == False then
14 |  |  return not_schedulable
15 |  end
16 end
17 return schedulable, priority
```

In the first *for* loop of Algorithm 3 (line 3), we iterate over priority levels in decreasing order. In the second *for* loop (line 5), we iterate over non assigned tasks (belonging to the set τ') and we check (line 6) if one of these tasks is feasible at the current priority level l when other non assigned tasks have higher priorities. Thus, we assign this task to level l (line 7) and we remove it from the set of non assigned tasks τ' (lines 8 and 9). If no task is assigned to (feasible at) a given priority

level (line 13), then no feasible priority ordering exists according to Theorem 5.1 and we return "*not_schedulable*" (line 14). Otherwise, we return "*schedulable*" and the priority ordering is found (line 17).

Algorithm 3 has two nested *for* loops calling the schedulability test at each iteration. For a task set composed of n tasks, this algorithm performs at most $(n^2 + n)/2$ schedulability tests in order to find a feasible priority ordering according to the schedulability test used, or to conclude that no such feasible priority ordering exists. Hence, we deduce that the complexity of Audsley's algorithm is $O(n^2 \times L)$ where L is the complexity of the schedulability test.

On the other hand, Davis and Burns [100, 101] generalize the use of Audsley's algorithm on multi-core (multiprocessor) platforms instead of only on a single core processor. They prove that this algorithm is an optimal priority assignment policy if the schedulability test S used by Audsley's algorithm, is compliant with the three conditions stated below. Consequently, Audsley's algorithm is also known as Optimal Priority Assignment (OPA). In this context, the term optimal is defined by Definition 1 in [101]. It means that if there is a priority ordering validating that a task set is schedulable according to the schedulability test S, then the priority ordering generated by Audsley's algorithm is also feasible according to the schedulability test S. In other words, the set Y of all task sets that are deemed schedulable by the schedulability test S using its optimal priority assignment policy (Audsley's algorithm) is a superset of the set Z ($Z \subseteq Y$) of all task sets that are deemed schedulable by test S using any other priority assignment policy.

These are the three conditions provided by Davis and Burns [100] that a schedulability test S used by an OPA policy should respect:

- **Condition 1:** The schedulability of a task τ_i may, according to test S, be dependent on the set of higher priority tasks, but not on the relative priority ordering of those tasks

- **Condition 2:** The schedulability of a task τ_i may, according to test S, be dependent on the set of lower priority tasks, but not on the relative priority ordering of those tasks.

- **Condition 3:** When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become non schedulable according to the test S, if it was previously schedulable at the lower priority.

In order to assign priorities at the task level for a DAG task model, we use Audsley's algorithm with the schedulability test based on our RTA proposed in Section 3.2.3.1 (Equations 3.21, 3.22 and 3.23). Therefore, we should prove that our RTA respects the previous conditions.

Proof. First, we prove that the response time of a DAG task τ_i obtained by our RTA is not affected by the relative order of higher priority DAG tasks. Indeed, the effect of all higher priority DAGs on τ_i is included in the global response time of the sink sub-task through the external interference term in Equation 3.23 (on Page 49). This external interference is given by Equation 3.24 where the sum term depends only on the set $hep(\tau_i)$ of higher priority DAGs and not on their relative priority ordering. Hence, the global response time depends only on higher priority DAGs and not on their relative priority ordering. Consequently, Condition 1 is verified.

Second, Condition 2 is obviously met in the case of preemptive scheduling with no blocking because lower priority DAG tasks cannot delay the execution of task τ_i. Thus, the response time of DAG task τ_i is not affected by the set of lower priority DAGs nor by their relative priority ordering.

Third, if a DAG task τ_i swaps its priority level with a higher priority DAG task, then the sum term in the external interference Equation 3.24 (on page 49) is reduced or it remains constant, because some elements in the set $hep(\tau_i)$ of higher priority DAGs may be removed. Thus, the global response time of τ_i remains the same or decreases but it cannot increase. Consequently, if τ_i was schedulable at its original priority level, then it cannot become non schedulable at the higher priority level.

Therefore, we deduce that the three conditions, proposed by Davis and Burns [100], are respected by our RTA. Hence, we could use this latter as a schedulability test for Audsley's algorithm while guaranteeing its optimality.

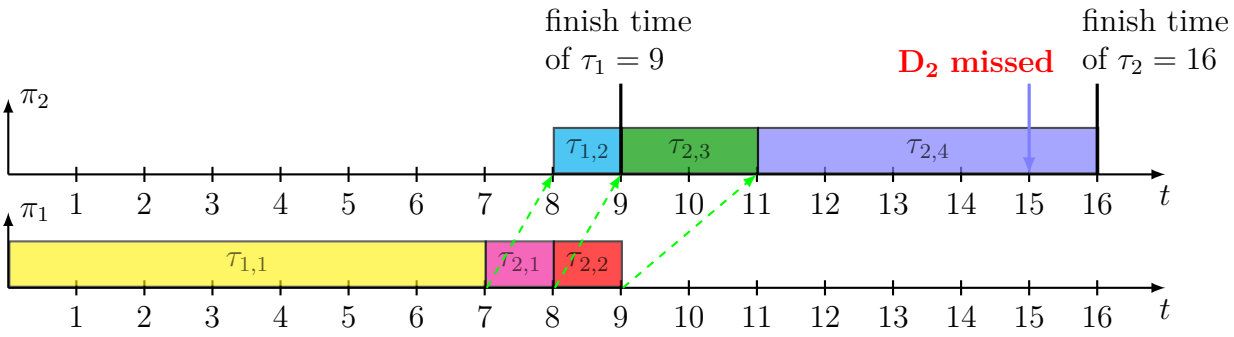
■

Example 5.2. *In this example, we use the same task set described by Figure 5.1 and Table 5.1 with some modified parameters. We change the execution times of sub-tasks $\tau_{1,1}$ and $\tau_{2,4}$ and the deadline of DAG task τ_1 as follows: $C_{1,1} = 7$ (instead of 9), $C_{2,4} = 5$ (instead of 2) and $D_1 = 18$ (instead of 11).*

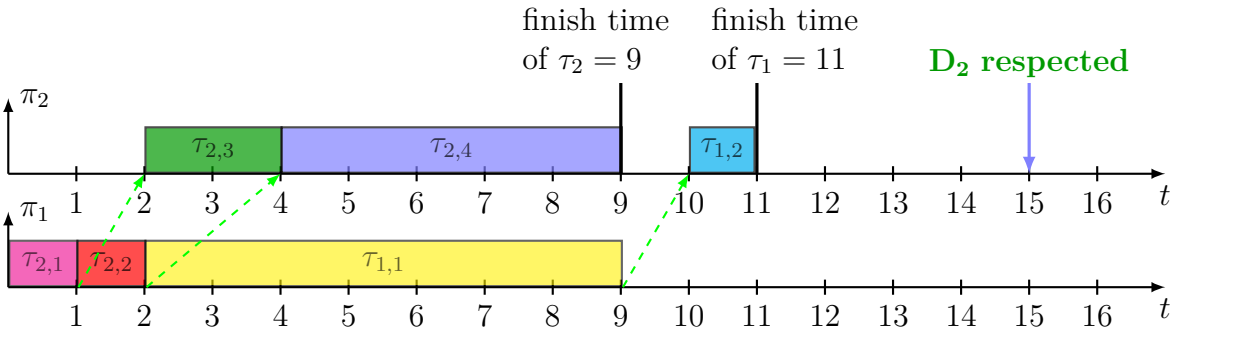
In the case where τ_2 has the lowest priority, its global response time according to our RTA is equal to $R^glob_2 = 17 > D_2$. Thus, τ_2 misses its deadline and any ordering that assigns to τ_2 the lowest priority is not feasible. However, if τ_1 has the lowest priority, then the computed global response times of the two DAGs are equal to $R^glob_1 = 18 < D_1$ and $R^glob_2 = 9 < D_2$. Hence, both DAGs respect their

deadlines and the ordering that assigns to τ_1 the lowest priority is feasible. We deduce that Audsley's algorithm provides a feasible priority ordering, according to our RTA, that assigns to τ_2 a higher priority than τ_1.

In Figure 5.3, we illustrate two different schedules for the previous task set with two different priority orderings of DAG tasks. We note that green dashed arrows represent communication delays between sub-tasks. In the first scheduling (Figure 5.3a), we assume that τ_1 has higher priority than τ_2. We note that DAG task τ_1 finishes its execution at time instant $t = 9$ before its deadline $D_1 = 18$. On the other hand, τ_2 finishes its execution at $t = 16$ and misses its deadline $D_2 = 15$ because it is delayed by sub-task $\tau_{1,1}$ on core π_1. In the second scheduling (Figure 5.3b), priorities are assigned according to Audsley's algorithm where τ_2 has higher priority than τ_1. In this case, we note that the two DAG tasks respect their deadline. Indeed, τ_1 finishes its execution at $t = 11 < D_1$ and τ_2 finishes its execution at $t = 9 < D_2$.



(a) First scheduling: τ_1 has higher priority than τ_2



(b) Second scheduling: τ_2 has higher priority than τ_1 (according to Audsley's algorithm)

Figure 5.3: Scheduling of task set defined in Figure 5.1 with two different priority assignments

5.1.2 Priority at the sub-task level

After assigning priorities at the task level, we focus on defining priorities for all sub-tasks inside each DAG task. This operation avoids ambiguity caused by arbitrary order of execution between parallel sub-tasks and helps to reduce the response time. For instance, two parallel sub-tasks having the same predecessors, are activated at the same time. Moreover, if they are mapped to the same core and have the same priority, then they will have an arbitrary order of execution. This may also increase the pessimism of schedulability analysis. To avoid such undefined behavior, we define priorities at the sub-task level. We determine these priorities for sub-tasks from the same DAG because they are already ordered with respect to other sub-tasks from other DAG tasks, based on the priority defined at the task level (Section 5.1.1). We note that assigning priorities for sub-tasks from the same DAG, transforms the partial order defined by precedence constraints to a total order that avoids an arbitrary order of execution between sub-tasks.

Moreover, the execution order of sub-tasks may promote or prevent parallel executions depending on the structure of the dependency graph and the core mapping. Therefore, the priority ordering of sub-tasks has an impact on (i.e. may increase or decrease) the response time of the whole DAG task. Hence, we define priorities for sub-tasks in such a way as to exploit possible parallelism and enhance the reactivity of the system.

In this part, we define priorities at the sub-task level. First, we give a scheduling example to highlight the importance of the sub-task priority ordering and its influence on DAG response time (Section 5.1.2.1). Second, we exploit the results of Baruah [59] about optimal scheduling of DAGs on partitioned processors based on an ILP formulation and we derive the optimal execution order and priority assignment for sub-tasks inside the same graph (Section 5.1.2.2). This approach has a high computational complexity since the multi-core task scheduling in general and the scheduling of DAGs are NP-hard problems [60, 102]. Third, we provide a heuristic algorithm with a polynomial complexity to approximate the best priority ordering of sub-tasks (Section 5.1.2.3). Finally, we apply a genetic algorithm based search to find a near optimal priority assignment for sub-tasks from the same DAG (Section 5.1.2.4).

5.1.2.1 Motivation Example

In order to illustrate the purpose of defining priority at the sub-task level, we schedule the DAG task defined by Figure 5.4 and Table 5.2 with two different sub-task priority orderings.

Example 5.3. *We consider the DAG task τ_1 (Figure 5.4) where sub-tasks $\tau_{1,1}, \tau_{1,2}$ and $\tau_{1,5}$ are mapped to the same core π_1 and the remaining sub-tasks are mapped to the second core π_2. For the sake of simplicity in this example, we assume that there are no communication delays between different sub-tasks of the DAG task τ_1. The minimum inter-arrival time of τ_1 is equal to $T_1 = 10$ and its deadline is $D_1 = 9$ (constrained deadline $D_1 \leq T_1$).*

From the structure of the DAG task τ_1 in Figure 5.4, we note that when sub-task $\tau_{1,1}$ finishes its execution, sub-tasks $\tau_{1,2}$ and $\tau_{1,5}$ are activated simultaneously on the core π_1. The first priority ordering ("Order 1" in Table 5.2), gives a higher priority to $\tau_{1,2}$ compared to $\tau_{1,5}$ while the second priority ordering ("Order 2" in Table 5.2) inverts their priorities.

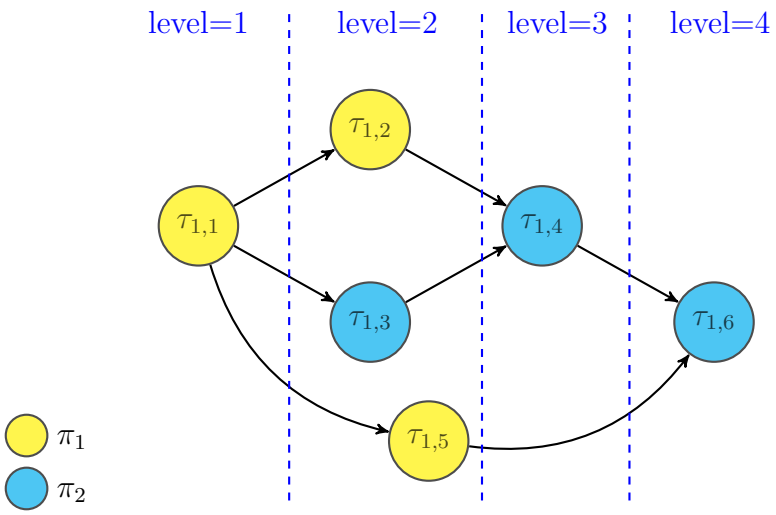
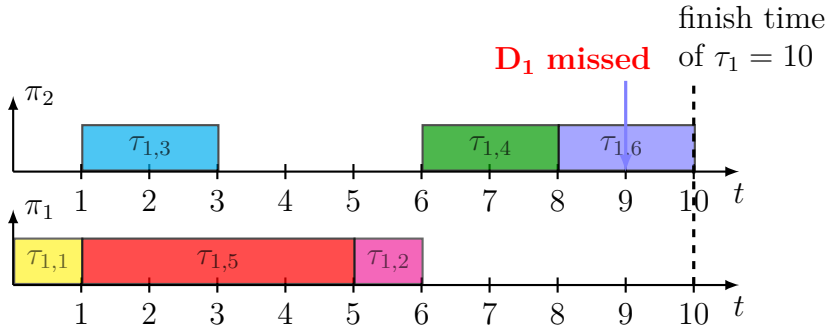


Figure 5.4: Example of a DAG task divided into several levels using topological ordering described by Kahn [103]

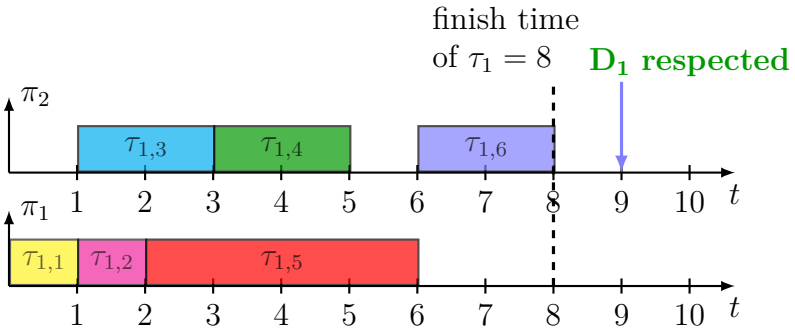
Table 5.2: Timing parameters and sub-task priority orderings of the DAG task described in Figure 5.4

Sub-task	$C_{i,j}$	T_i	D_i	Core	Order 1	Order 2
$\tau_{1,1}$	1	10	9	π_1	1	1
$\tau_{1,2}$	1	10	9	π_1	**2**	**4**
$\tau_{1,3}$	2	10	9	π_2	3	3
$\tau_{1,4}$	2	10	9	π_2	5	5
$\tau_{1,5}$	4	10	9	π_1	**4**	**2**
$\tau_{1,6}$	2	10	9	π_2	6	6

In Figure 5.5, we illustrate the two schedulings of DAG task τ_1 corresponding to the two priority orderings in Table 5.2. In the first scheduling (Figure 5.5a), sub-task $\tau_{1,5}$ is executed before $\tau_{1,2}$. We note that the response time of the whole DAG τ_1 is equal to $R_1^{global} = 10$. On the other hand, if $\tau_{1,2}$ is executed before $\tau_{1,5}$ as in the second scheduling (Figure 5.5b), the response time of τ_1 is reduced $R_1^{global} = 8$.



(a) First scheduling: $\tau_{1,5}$ has higher priority than $\tau_{1,2}$ ($R_1^{global} = 10$)



(b) Second scheduling: $\tau_{1,2}$ has higher priority than $\tau_{1,5}$ ($R_1^{global} = 8$)

Figure 5.5: Examples of scheduling of the DAG task defined in Figure 5.4 according to different sub-task orderings.

We conclude that the priority ordering of sub-tasks could influence the response time of a DAG task by promoting or preventing the exploitation of possible parallelism derived from the DAG structure and the mapping to the existing cores.

5.1.2.2 Optimal Sub-task Priority Assignment

We aim to assign priorities for sub-tasks in a way that reduces the response time of the whole DAG. To do so, we are inspired by a recent work of Baruah [59]. In this work, the author proposes an MILP formulation of the scheduling problem of partitioned DAG tasks. He uses techniques from operations research literature like ordering variables. Then, he combines them with an approach from the real-time scheduling domain, which is the optimal schedulability test for EDF (Earliest Deadline First) scheduling on a uniprocessor based on the demand bound function [15].

As stated in [59], the size of the MILP problem is polynomial in the number of sub-tasks n_i. In fact, the MILP formulation requires $O(n_i^3)$ non-negative real-valued variables that represent the start and finish times of each sub-task and other variables used for the computation of the demand bound function for each triplet of sub-tasks. This formulation also uses $O(n_i^2)$ zero-one variables that define the execution order between each pair of sub-tasks. Each of these variables has a constant number of linear constraints, thus this MILP introduces $O(n_i^3)$ constraints.

We adjust the constraints applied on ordering variables in [59] in order to model more general cases of the DAG scheduling problem. More details about these constraints are given in Appendix A. After modifying these constraints, we use an optimization solver [104, 105] to minimize the finish time of the sink sub-task and accordingly the response time of the whole DAG. Thus, we obtain an optimal scheduling for the DAG task under study, defined by the start and finish time of each sub-task. In some cases, these start and finish times define the same interval for several sub-tasks to execute successively on the same core without clearly determining the execution order of these sub-tasks.

We run EDF scheduling algorithm for each core on the set of sub-tasks assigned to that core and characterized by individual release times and deadlines equal to the start and finish times respectively. Consequently, we obtain the exact scheduling interval of each sub-task. In order to determine the priority ordering for sub-tasks, we sort them according to their exact start times and in the case of equality for several sub-tasks on different cores, we prioritize the one with the lowest finish time.

Despite the polynomial size of the MILP formulation, the scheduling problem of DAG tasks remains an NP-hard problem as stated by Baruah [59]. In the remainder of this section, we propose two approximation methods (heuristic based and evolutionary based) that, in practice, provide solutions relatively close to the optimal one while reducing the computational complexity.

5.1.2.3 Sub-task Priority Assignment Heuristic

In Example 5.3, we show that the execution order of sub-tasks may reduce the response time of the whole DAG. Finding the optimal priority ordering of sub-tasks that leads to the lowest response time of the whole DAG, has a high computation complexity as stated in Section 5.1.2.2. Hence, we propose a heuristic algorithm with a polynomial complexity in the number of sub-tasks, but it does not guarantee finding the optimal ordering.

Our algorithm assigns priorities to the sub-tasks based on their successors workload. The successors workload of a sub-task $\tau_{i,j}$ that run on different cores is denoted by $succ\_sum(\tau_{i,j})$ and is defined as follows:

$$succ\_sum(\tau_{i,j}) = \sum_{\substack{\tau_{i,k} \in succ(\tau_{i,j}) \\ \pi(\tau_{i,k}) \neq \pi(\tau_{i,j})}} C_{i,k} \tag{5.1}$$

Our algorithm prioritizes a sub-task $\tau_{i,j}$ if it has the maximum successors workload that run on cores different from the one of $\tau_{i,j}$ (see Equation 5.1). Indeed, when such sub-task completes its execution earlier, it allows successors mapped to other cores to start their execution earlier and probably to run in parallel with successors mapped to the same core. For instance, in Figures 5.4 and 5.5, when the sub-task $\tau_{1,2}$ is executed before $\tau_{1,5}$, it allows $\tau_{1,4}$ to start its execution earlier on the other core π_2 and consequently to run in parallel with $\tau_{1,5}$.

In the case of equality between two sub-tasks or more according to the first criteria (i.e. successors workload), we resort to a second criteria based on topological ordering described by Kahn [103]. In fact, we split the DAG into several levels that respect precedence constraints (see the example in Figure 5.4) and we prioritize the sub-task that belongs to a prior level. This strategy gives higher priority to a predecessor sub-task than its successors, which is consistent the precedence constraints and their corresponding partial order.

Algorithm 4 illustrates how the priority assignment heuristic works. First, we initialize *succ_sum* vector (line 1) and we calculate, for each sub-task $\tau_{i,j}$, the sum of the execution time of successors $succ(\tau_{i,j})$ that are executed on different cores than $\tau_{i,j}$, i.e. $\pi(\tau_{i,k}) \neq \pi(\tau_{i,j})$ (lines $2-8$). Then, we separate sub-tasks of τ_i into levels (line 9) using topological ordering[103]. Finally, we sort sub-tasks (line 10) in decreasing order of the sum of successors workload (*succ_sum*). We break a tie by selecting the sub-task with the lowest level.

Algorithm 4: Priority assignment for sub-tasks belonging to the same DAG

Data: τ_i a DAG task, n_i number of sub-tasks in τ_i and $\pi(\cdot)$ the core mapping
Result: Priority ordering of sub-tasks

```
succ_sum = zeros(n_i) /* Intialized vector of size n_i */
for τ_{i,j} ∈ τ_i do
    for τ_{i,k} ∈ succ(i,j) do
        if π(τ_{i,k}) ≠ π(τ_{i,j}) then
            succ_sum(τ_{i,j}) = succ_sum(τ_{i,j}) + C_{i,k}
        end
    end
end
levels = topologic_order(τ_i)
Priority = argsort(τ_i, order = [−succ_sum, levels])
return Priority
```

Example 5.4. *In this example, we apply the priority assignment heuristic (Algorithm 4) on the DAG task defined in Figure 5.4 and Table 5.2. The results are presented in Table 5.3.*

Table 5.3: Applying priority assignment heuristic for sub-tasks on the DAG task described in Figure 5.4

Sub-task	$C_{i,j}$	succ($\tau_{i,j}$)	Core	succ_sum	Priority
$\tau_{1,1}$	1	$\{\tau_{1,2}, \tau_{1,3}, \tau_{1,4}, \tau_{1,5}, \tau_{1,6}\}$	π_1	6	1
$\tau_{1,2}$	1	$\{\tau_{1,4}, \tau_{1,6}\}$	π_1	**4**	**2**
$\tau_{1,3}$	2	$\{\tau_{1,4}, \tau_{1,6}\}$	π_2	0	4
$\tau_{1,4}$	2	$\{\tau_{1,6}\}$	π_2	0	5
$\tau_{1,5}$	4	$\{\tau_{1,6}\}$	π_1	**2**	**3**
$\tau_{1,6}$	2	$\varnothing$	π_2	0	6

We note that sub-task $\tau_{1,2}$ has a higher successors workload (succ_sum) than $\tau_{1,5}$. Hence, our heuristic assigns to $\tau_{1,2}$ a higher priority than $\tau_{1,5}$. Moreover, sub-tasks $\tau_{1,3}$, $\tau_{1,4}$ and $\tau_{1,6}$ have the same successors workload that is equal to zero. In this case, our algorithm resorts to the second criteria and assigns the higher priority to the sub-task belonging to the prior level (see Figure 5.4). Thus, $\tau_{1,3}$ has a higher priority than $\tau_{1,4}$, which has a higher priority than $\tau_{1,6}$.

In order to evaluate the complexity of our proposed Algorithm 4, we consider its components as follows:

- The two nested *for* loops compute the successors workload (*succ_sum*) using a simple sum operation. They have a complexity equal to $O(n_i^2)$, where n_i is the number of sub-tasks in the DAG task τ_i.

- The *topologic_order* function used in line 9 implements the topological ordering described by Kahn [103]. Its complexity equal to $O(n_i + \mid E_i \mid)$, where $\mid E_i \mid$ is the number of edges in the DAG. This number is bounded from above by n_i^2. Thus, the complexity of *topologic_order* function is also upper bounded by $O(n_i^2)$.

- The $argsort$ function used in line 10 has a complexity that is equal to $O(n_i \times \log n_i)$.

We deduce that the complexity of the entire priority assignment heuristic is equal to $O(n_i^2)$. Hence, it runs in polynomial time in the number of sub-tasks n_i.

On the other hand, it is clear that the proposed heuristic is an *m-approximation* algorithm. It provides a solution that cannot exceed the optimal solution times m, where m is the number of cores. In fact, the response time R_i^H of the DAG τ_i obtained with the priority ordering generated by our heuristic, is bounded by the cumulative execution time of all sub-tasks i.e. $R_i^H \leq \sum_{\tau_{i,j} \in \tau_i} C_{i,j}$ (i). On the other hand, the response time $R_i^\star$ corresponding to the optimal ordering is always greater than the cumulative execution time of all sub-tasks divided by the number of cores m i.e. $\sum_{\tau_{i,j} \in \tau_i} C_{i,j}/m \leq R_i^\star$ (ii). By multiplying (i) and (ii), we obtain $\frac{R_i^H}{m} \leq R_i^\star$. Thus, we deduce that R_i^H cannot exceed $R_i^\star \times m$.

Remark. *In the case of probabilistic task model, we use the expected value* $\mathbb{E}(\mathcal{C}_{i,j})$ *of execution time distribution* $\mathcal{C}_{i,j}$ *instead of the deterministic value* $C_{i,j}$ *to compute* $succ\_sum(\tau_{i,j})$*. We proceed similarly for all the algorithms in this chapter by using expected values instead of deterministic values in order to deal with probabilistic parameters.*

5.1.2.4 Sub-task Priority Assignment with a Genetic Algorithm

In this part, we build another sub-task priority assignment algorithm that offers near optimal parallel executions on different processors based on a Genetic Algorithm (GA) [106]. Genetic algorithms are inspired by natural evolution theory. They mainly consist of evolving a group of possible solutions called a population by successively selecting the best members and combining them to create new members that are likely to be good.

In our case, we consider a population composed of a set of possible priority assignments of sub-tasks inside the same graph. Each member in the population is a vector of size n_i (the number of sub-tasks in DAG τ_i) where each element represents a priority level in descending order and it contains the sub-task assigned to that priority level (see the example in Figure 5.8).

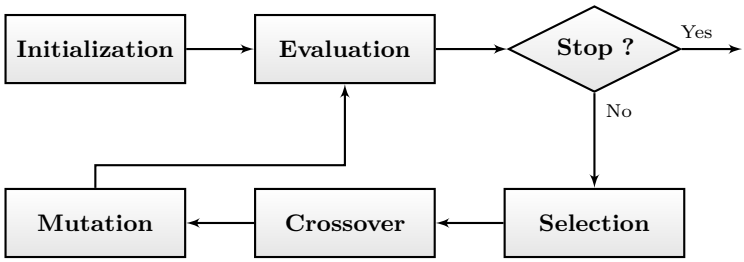


Figure 5.6: Flowchart of the proposed Genetic Algorithm

Our proposed GA-based heuristic is an iterative procedure composed of several steps (cf. Figure 5.6) detailed as follows:

- **Initialization:** In order to initialize population members, we use random priority ordering derived from topological order [103] and we generate several possible priority assignments for sub-tasks composing the DAG under study.

- **Evaluation:** In this step, we evaluate the objective function (fitness) of each member of the population. To do so, we compute the response time of the studied graph while considering the priority assignment that corresponds to the population member evaluated.

- **Selection:** We select the two best members, called the winner and the loser, that have priority assignments corresponding to the two least response times in the population. Then, we keep the winner in the next generation (iteration) and we replace the loser by the child member. The latter is built by applying evolution operations (crossover and mutation) on the pair composed of the winner and the loser members. The elitist selection preserves the best member in the next generation and prevents the degradation of the population fitness.

The proposed GA is based on two main evolution operations:

- **Crossover** (cf. Figure 5.8): We select the priority order of a subset of nodes from the loser and insert it in the winner while respecting the topological order to obtain the child member.

- **Mutation** (cf. Figure 5.9): We swap the priorities of two parallel sub-tasks from the child member while respecting the topological order.

For the stop condition, we use a limited number of iterations equal to 100 because we note that beyond this number, the winner response time becomes almost constant. The winner of the last generation presents the priority assignments that most reduce the response time of the DAG task.

Example 5.5. *Let us consider an example of one iteration of the deployed GA. Figure 5.7 presents the DAG task, and we aim to determine a priority ordering for its sub-tasks.*

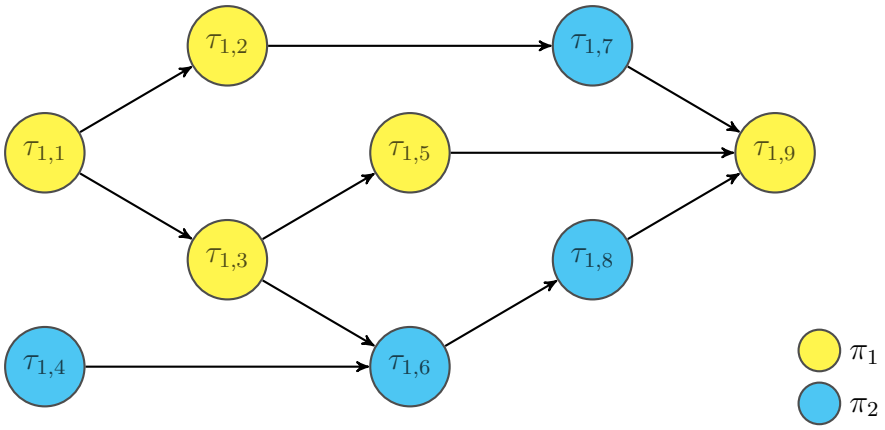


Figure 5.7: Example of a DAG task τ_1 to determine its sub-tasks priorities

First, we randomly initialize a population. Each member is presented as a sequence of sub-tasks sorted in descending order of priority while respecting the sub-tasks dependencies (topological order of the DAG). After computing the response time of all the members, we select the two best members of the population called the winner and the loser (cf. Figure 5.8).

During the crossover operation (Figure 5.8), we select two parallel sub-tasks having different relative orders in the winner and the loser ($\tau_{1,2}$ and $\tau_{1,3}$). As the winner has the best fitness, the child member is built by conserving most of the winner's sequence and changing the order of the two selected sub-tasks according to the loser ordering ($\tau_{1,3}$ has higher priority than $\tau_{1,2}$ in the child member as in the loser). This priority change should respect the precedence constraints. Therefore, it induces a possible change in the priorities: (i) of the predecessors ($\tau_{1,1}$) of the selected sub-task with the lower priority in the winner ($\tau_{1,3}$) and (ii) of the successors ($\tau_{1,7}$ and $\tau_{1,9}$) of the selected sub-task with the higher priority in the winner ($\tau_{1,2}$).

Then, we apply the mutation operation (Figure 5.9). We swap the priorities of two parallel sub-tasks ($\tau_{1,3}$ and $\tau_{1,4}$) from the child member while respecting the

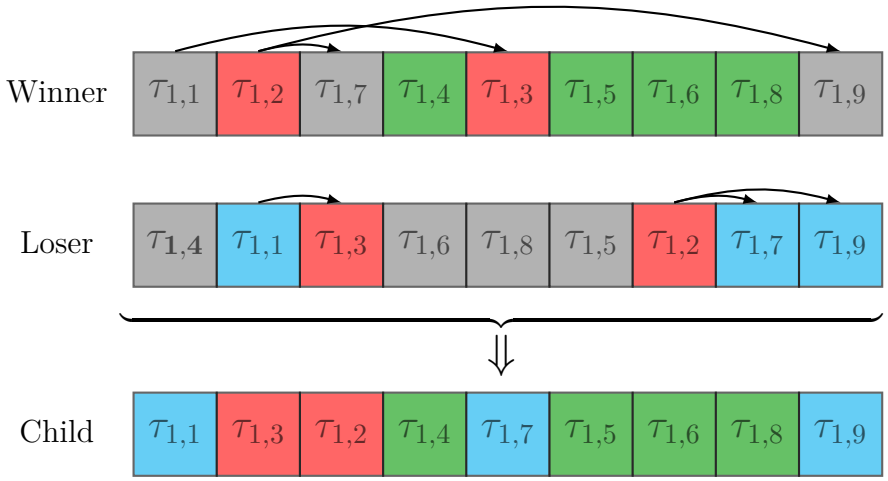


Figure 5.8: Crossover operation of genetic algorithm

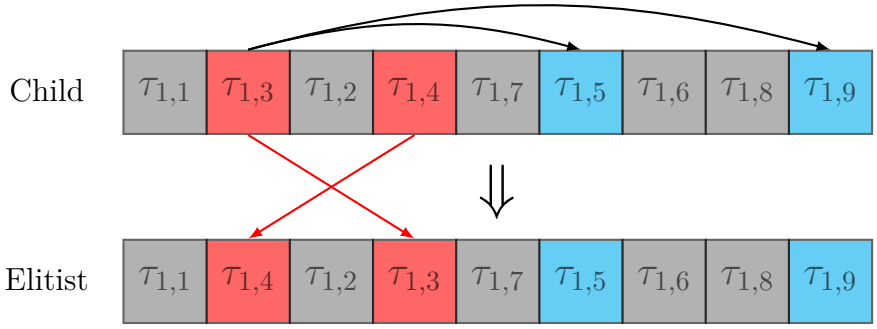


Figure 5.9: Mutation operation of genetic algorithm

precedence constraints. We replace the loser in the previous population by the new child member (Elitist) and we iterate until reaching the stop condition.

5.2 Partitioning Heuristic

Several categories of partitioning algorithms that map executing programs (task, sub-task) to existing processors or cores have been studied in the literature. Some heuristics originally devised for the bin packing problem, are used for allocating independent and periodic tasks with recurrent jobs. From this category of partitioning heuristics, we list Next Fit (NF) [107], First Fit (FF) [108], Best Fit (BF) [109], and Worst Fit (WF) [108]. These algorithms allocate tasks to processors based on their decreasing order of utilization and aim to minimize the number of processors required.

Other partitioning heuristics are based on a list scheduling approach. They mainly operate on dependent tasks and they allocate each node of a DAG task to one processor and schedule them while respecting the precedence constraints. The

basic idea of these approaches is to make a list of nodes to be scheduled and to sort them using a priority ordering technique like Highest level First (HLF) [110], Longest Path (LP) [111] or Critical Path (CP) [112].

The second step of list scheduling is the processor selection. It consists in selecting a node from the scheduling list according to the defined order and allocating it to the processor that minimizes its start time. This step is repeated until all the nodes have been scheduled. The computation of the start time of a given node on a given processor, is based on whether a non-insertion or an insertion policy is adopted [113]. The latter considers the possible insertion of a node in a prior idle time slot on a processor when computing the earliest start time. The idle time slot considered should come after the activation time of the node in order to preserve precedence constraints.

In general, partitioning algorithms based on list scheduling are used for a single DAG task. They could also be applied on multiple DAG tasks that all have the same period by considering them as a single non-connected graph. Moreover, we could apply these partitioning heuristics on multiple DAG tasks with different periods by unfolding their executions to the hyperperiod in order to obtain mono-rate DAG tasks similarly to the approach used in [58]. However, the hyperperiod could be potentially large which may explode the number of nodes and increase the complexity. To the best of our knowledge, partitioning algorithms in the literature do not deal directly with a DAG task model with individual periods for each DAG.

In this section, we propose a partitioning heuristic that operates on several DAG tasks with different periods. It takes into consideration the structure of dependencies graphs as well as their individual periods, utilizations and priorities. We assume that the priorities at the task level are given. In general, communication delays between sub-tasks inside the same DAG become significant when two related sub-tasks are mapped to different cores. Hence, we consider that the communication delay between two sub-tasks is equal to zero if the they are mapped to the same core. Our partitioning heuristic aims to minimize introduced communication delays while balancing the load between cores in a way that maximizes possible parallelism.

Algorithm 5 illustrates how our partitioning heuristic works. We process DAG tasks in descending order of their priorities (line 2). Then, we consider the sub-tasks inside each DAG task in ascending order of levels (line 4). These levels are determined by topological ordering [103] as previously explained in Section 5.1.2.3 (see example in Figure 5.4). For each core π_k, we compute the cost function $F_{cost}(\tau_{i,j}, \pi_k)$ defined in the following Equation 5.2:

$$F_{cost}(\tau_{i,j}, \pi_k) = \frac{C_{i,j}}{1 - U_{\pi_k}} + \sum_{\substack{\tau_{i,l} \in ipred(\tau_{i,j}) \\ \pi(\tau_{i,l}) \neq \pi(\tau_{i,j})}} \frac{e_i(l,j)}{1 - U_{com}} \tag{5.2}$$

We denote by U_{π_k} the utilization of the core π_k. It is equal to the sum of the utilization of each sub-task mapped to π_k. Similarly, U_{com} refers to the utilization of a virtual core that represents the communication bus. It is equal to the sum of the utilization of each introduced communications delay.

The cost function $F_{cost}(\tau_{i,j}, \pi_k)$ (Equation 5.2) estimates the average response time of sub-task $\tau_{i,j}$ if it is mapped to the core π_k. Then, $\tau_{i,j}$ is assigned to the core π_{best} that minimizes this cost function (lines $5 - 12$ in Algorithm 5). After that, we update the utilization of π_{best} by adding the utilization of the sub-task $\tau_{i,j}$ (line 13) and we also update the communication utilization by the new introduced communication delays caused by the predecessors that are mapped to different cores (lines $14 - 18$).

Algorithm 5: Sub-tasks partitioning algorithm

Data: τ set of n DAG task and π set of m cores
Result: Sub-tasks mapping $\pi(\tau_{i,j})$

```
1  U = zeros(m + 1) /* m core utilizations + 1 communication
       utilization                                          */
2  for τ_i ∈ τ (in descending order of task priority) do
3      levels = topologic_order(τ_i)
4      for τ_{i,j} ∈ τ_i (in ascending order of levels) do
5          min_cost = +∞
6          for π_k ∈ π do
7              if F_cost(τ_{i,j}, π_k) < min_cost and U[π_k] + C_{i,j}/T_i ≤ 1 then
8                  min_cost = F_cost(τ_{i,j}, π_k)
9                  π_best = π_k
10             end
11         end
12         π(τ_{i,j}) = π_best
13         U[π(τ_{i,j})] = U[π(τ_{i,j})] + C_{i,j}/T_i
14         for τ_{i,l} ∈ ipred(τ_{i,j}) do
15             if π(τ_{i,l}) ≠ π(τ_{i,j}) then
16                 U[com] = U[com] + e_i(l,j)/T_i
17             end
18         end
19     end
20 end
21 return π(τ_{i,j}) sub-tasks mapping for all τ_{i,j}
```

In order to evaluate the complexity of proposed partitioning Algorithm 5, we consider its components as follows:

- The inner *for* loop (line 6) computes the cost function $F_{cost}(\tau_{i,j}, \pi_k)$ for each core π_k of the m available cores. The computational complexity of the cost function equal to $O(n_i)$, where n_i is the number of sub-tasks in the DAG. This complexity is derived from the sum term in Equation 5.2. Hence, the complexity of the *for* loop (line 6) is $O(m \times n_i)$.

- The complexity of the other inner *for* loop (line 14) is $O(n_i)$

- The two outer *for* loops examine all the sub-tasks in the task set. The total number of these sub-tasks is $N_{\text{sub-task}} = \sum_i n_i$.

We deduce that the complexity of the partitioning heuristic is equal to $O(N_{\text{sub-task}} \times m \times \max_i n_i)$. Hence, it is is polynomial in the total number of sub-tasks $N_{\text{sub-task}}$ and in the number of cores m.

Example 5.6. *In this example, we apply the partitioning Algorithm 5 of the task set defined in Figure 5.10 on a processor with three cores (i.e.* $\pi = \{\pi_1, \pi_2, \pi_3\}$*). This task set is composed of two DAG tasks* τ_1 *and* τ_2*. Their respective periods are equal to* $T_1 = 20$ *and* $T_2 = 40$*. The execution times of all sub-tasks are given in Table 5.4. We assume that* τ_1 *has higher priority than* τ_2*. For the sake of simplicity in this example, we consider that the communication delay between two sub-tasks mapped to different cores, is equal to* 1 *time unit.*

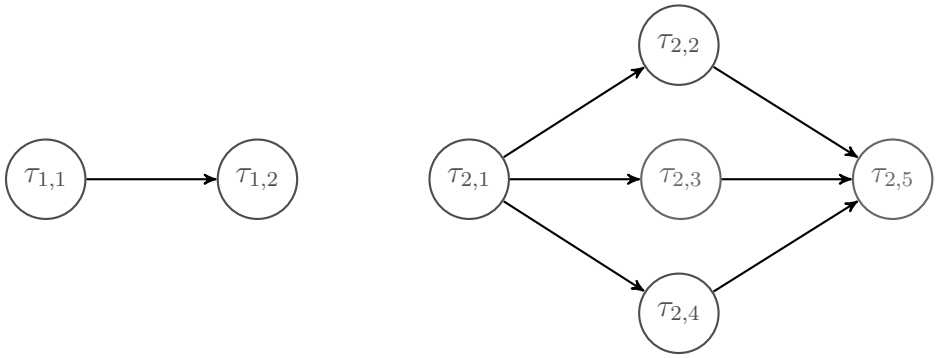


Figure 5.10: Example of a DAG task with sub-tasks not mapped to cores

Table 5.4: Execution of partitioning Algorithm 5 on the DAG task defined in Figure 5.10

Sub-task	$\mathbf{C_{i,j}}$	$\mathbf{F_{cost}}(\tau_{\mathbf{i,j}}, \pi_{\mathbf{1}})$	$\mathbf{F_{cost}}(\tau_{\mathbf{i,j}}, \pi_{\mathbf{2}})$	$\mathbf{F_{cost}}(\tau_{\mathbf{i,j}}, \pi_{\mathbf{3}})$	$\pi(\tau_{\mathbf{i,j}})$	$\mathbf{U_1}$	$\mathbf{U_2}$	$\mathbf{U_3}$	$\mathbf{U_{com}}$
$\tau_{1,1}$	2	**2**	2	2	π_1	0.1	0	0	0
$\tau_{1,2}$	1	**1.11**	2	2	π_1	0.15	0	0	0
$\tau_{2,1}$	4	4.7	**4**	4	π_2	0.15	0.1	0	0
$\tau_{2,2}$	8	10.41	**8.88**	9	π_2	0.15	0.3	0	0
$\tau_{2,3}$	7	9.23	10	**8**	π_3	0.15	0.3	0.175	0.025
$\tau_{2,4}$	8	**10.43**	11.42	10.72	π_1	0.35	0.3	0.175	0.05
$\tau_{2,5}$	4	6.15	5.71	**4.84**	π_3	0.35	0.3	0.275	0.1

Each row in Table 5.4, represents one iteration of the partitioning Algorithm 5 applied on the sub-task $\tau_{i,j}$. We compute the cost function $F_{cost}(\tau_{i,j}, \pi_k)$ of assigning $\tau_{i,j}$ to each of the three cores (Columns $3-5$ in Table 5.4). The minimum cost is highlighted and the corresponding core is assigned to $\tau_{i,j}$ (Column 6). After that, we update the corresponding utilization (Columns $7-9$). If a communication delay is introduced due to the mapping of $\tau_{i,j}$ to a different core than its predecessors, we also update the communication utilization U_{com}.

We note that the utilization of the three cores has values around 0.3 confirming that our algorithm balances the load between cores. Moreover, the communication utilization is equal to 0.1 due to the trade-off between parallelizing and reducing communication between cores.

5.3 Graph Reduction

As mentioned previously, the complexity of schedulability analysis depends on the number of DAG tasks as well as on the number of nodes in each DAG. For instance, the proposed schedulability test for probabilistic task sets based on Bayesian inference (in Section 4.3), has an exponential time complexity in the number of sub-tasks composing each DAG.

In this section, we propose to reduce the number of nodes inside a DAG task without affecting the precedence constraints defined by the original DAG. Our idea consists in merging connected sub-tasks that are executed on the same core and replacing them by a single sub-task with an execution time equal to the sum of all merged sub-tasks. In some cases, this merge operation imposes new precedence constraints between the new sub-task (i.e. the merged sub-tasks composing it) and other sub-tasks. Therefore, we should ensure that no additional precedence constraints are added by the merge operation.

For instance, in Figure 5.11, we note that by merging the sub-tasks $\tau_{1,7}$ and $\tau_{1,8}$ there are no new precedence constraints added. This is because the new node successors (i.e. $\tau_{1,9}$) are also successors of the nodes composing it (i.e. $\tau_{1,7}$ and $\tau_{1,8}$). The same also applies for the predecessors. Thus, these two nodes could be fused without modifying the structure of the precedence constraints. However, if we consider merging sub-tasks $\tau_{1,1}$ and $\tau_{1,2}$ in Figure 5.11, a precedence constraint from $\tau_{1,1}$ to $\tau_{1,5}$ will be added since $\tau_{1,5}$ is a successor of one of the merged sub-tasks (i.e. $\tau_{1,2}$) but not of the others (i.e. $\tau_{1,1}$).

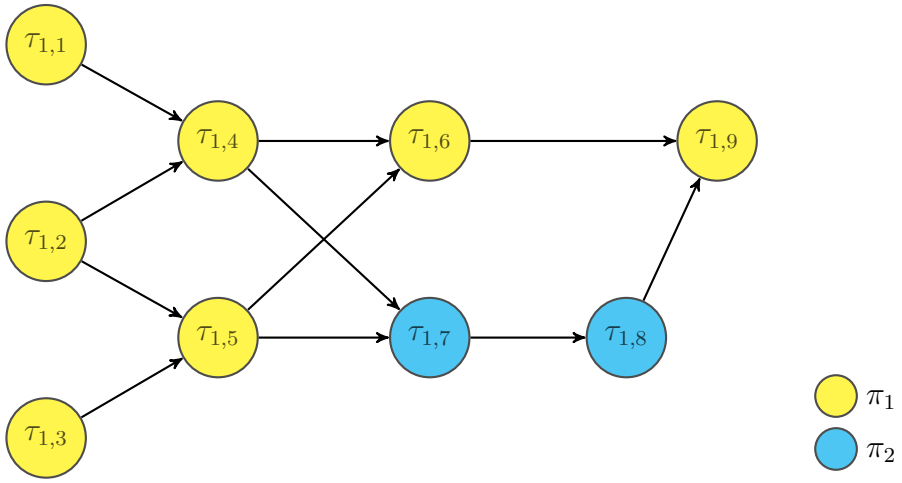


Figure 5.11: Example of a DAG task with sub-tasks assigned to different cores

In order to preserve the same structure of precedence constraint when merging sub-tasks together, the set Γ of merged sub-tasks that are mapped to the same core should satisfy these two conditions:

- **Condition 1:** If a sub-task $\tau_{i,l}$ is a predecessor of one of the sub-tasks in the set Γ and if $\tau_{i,l}$ is not in Γ, then $\tau_{i,l}$ should also be a predecessor for all sub-tasks in the set Γ.

- **Condition 2:** If a sub-task $\tau_{i,l}$ is a successor of one of the sub-tasks in the set Γ and if $\tau_{i,l}$ is not in Γ, then $\tau_{i,l}$ should also be a successor for all sub-tasks in the set Γ.

More formally, we define the set Γ of mergeable sub-tasks as follows:

Definition 5.1. *Let τ_i be a DAG task. A set Γ of sub-tasks from τ_i is said to be a "mergeable" set if for each pair of sub-tasks $\tau_{i,j}$ and $\tau_{i,k}$ that belong to Γ, we have:*

- $\pi(\tau_{i,j}) = \pi(\tau_{i,k})$

- $pred(\tau_{i,j}) \setminus \Gamma = pred(\tau_{i,k}) \setminus \Gamma$
- $succ(\tau_{i,j}) \setminus \Gamma = succ(\tau_{i,k}) \setminus \Gamma$

We note that for a DAG task τ_i there are several mergeable sets that could be disjoint, overlapping or included one in the other. Thus, we define a "maximal mergeable" set as follows:

Definition 5.2. *A set of sub-tasks Γ^{max} is said to be a "maximal mergeable" set if it is a meargeable set that cannot be a subset of any other mergeable set.*

We note that a sub-task $\tau_{i,j}$ could belong to different mergeable sets of sub-tasks. However, we prove that it belongs to only one maximal mergeable.

Theorem 5.2. *Let τ_i be a DAG task partitioned on m cores. Each sub-task $\tau_{i,j}$ belongs to only one maximal mergeable set denoted $\Gamma^{max}_{i,j}$.*

Proof. We prove this theorem by contradiction. Let us assume that there are two distinct maximal mergeable sets $\Gamma^1_{i,j}$ and $\Gamma^2_{i,j}$ that include a sub-task $\tau_{i,j}$. $\Gamma^1_{i,j}$ is a mergeable set. Thus, all sub-tasks in $\Gamma^1_{i,j}$ are executed on the same core as $\tau_{i,j}$ (i.e. the core $\pi(\tau_{i,j})$). If a successor of $\tau_{i,j}$ is not in $\Gamma^1_{i,j}$, then it is a successor of all sub-tasks in $\Gamma^1_{i,j}$. Similarly, for sub-tasks in $\Gamma^2_{i,j}$. Hence, we deduce that if a successor of $\tau_{i,j}$ is not in $\Gamma^1_{i,j}$ nor in $\Gamma^2_{i,j}$, then it is a successor of all sub-tasks in $\Gamma^1_{i,j} \cup \Gamma^2_{i,j}$.

Applying a similar reasoning for predecessors, we deduce that $\Gamma^1_{i,j} \cup \Gamma^2_{i,j}$ is a meargable set that contains $\tau_{i,j}$. This is contradictory to the fact that $\Gamma^1_{i,j}$ is a maximal mergeable set and cannot be a subset of any other mergeable set.

We conclude that $\Gamma^1_{i,j}$ and $\Gamma^2_{i,j}$ could not be distinct and there is only one maximal mergeable set $\Gamma^{max}_{i,j}$ that contains a sub-task $\tau_{i,j}$.

■

Remark. *Let $\Gamma^{max}_{i,j}$ and $\Gamma^{max}_{i,k}$ be two maximal mergeable sets corresponding to sub-tasks $\tau_{i,j}$ and $\tau_{i,k}$ respectively. Therefore, they cannot overlap and they are either disjoint or equal if one sub-task belongs to the maximal mergeable set of the other (i.e. $\tau_{i,j} \in \Gamma^{max}_{i,k}$ or $\tau_{i,k} \in \Gamma^{max}_{i,j}$).*

If a sub-task $\tau_{i,j}$ is not mergeable with any other sub-tasks that are executed on the same core, then $\Gamma^{max}_{i,j} = \{\tau_{i,j}\}$.

In order to reduce the number of nodes in a partitioned DAG task to the minimum, we should find the maximal mergeable sets for each node. Then, we merge sub-tasks in each of these sets. In the remainder of this section, we propose two methods to determine the maximal mergeable sets and to reduce the size of a DAG. The first one, based on an ILP formulation, enables us to optimally reduce the DAG. The second method is a greedy heuristic. It iteratively merges any two mergeable nodes that respect the three conditions in Definition 5.1, until no further merging is possible.

5.3.1 ILP based approach

Let G be a DAG graph composed of n nodes $v_i,\ \forall\ i \in \{1, \ldots, n\}$ and partitioned on m cores. In this part, we propose an ILP formulation to represent the problem of determining the maximal mergeable set Γ_i^{max} of the node v_i. This ILP based approach is applied for each node in order to reduce the whole graph G.

First, we define for a studied DAG G the dependencies matrix Δ such that:

$$\Delta_{j,k} = \begin{cases} 1 & \text{if } v_k \in succ(v_j) \\ 0 & \text{otherwise} \end{cases}$$

We note that $\Delta_{j,j} = 0,\ \forall\ j \in \{1, \ldots, n\}$.

We also define n zero-one integer variables $x_j,\ \forall\ j \in \{1, \ldots, n\}$ such that:

$$x_j = \begin{cases} 1 & \text{if the node } v_j \text{ is inclued in the maximal} \\ & \text{mergeable set } \Gamma_i^{max} \text{ of the node } v_i \\ 0 & \text{otherwise} \end{cases}$$

Since all the nodes in the maximal mergeable set Γ_i^{max} should be mapped to the same core as v_i, we impose as a constraint $x_j = 0$ if $\pi(v_j) \neq \pi(v_i),\ \forall\ j \in \{1, \ldots, n\}$.

According to "Condition 1" stated above, for each two nodes v_j and v_k, belonging to the maximal mergeable set Γ_i^{max}, if a node $v_l \notin \Gamma_i^{max}$ is a predecessor of v_j, then v_l should also be a predecessor of v_k. We present this condition using integer variables of the ILP problem as follows:

$$(1 - x_l) \cdot x_j \cdot x_k \cdot \Delta_{l,j} \leq \Delta_{l,k} \quad \forall\ j \neq l,\ k \neq l \tag{5.3}$$

According to "Condition 2" stated above, For each two nodes v_j and v_k belonging to the maximal mergeable set Γ_i^{max}, if a node $v_l \notin \Gamma_i^{max}$ is a successor of v_j, then v_l should also be a successor of v_k.

$$(1 - x_l) \cdot x_j \cdot x_k \cdot \Delta_{j,l} \leq \Delta_{k,l} \quad \forall\ j \neq l,\ k \neq l \tag{5.4}$$

The two constraints defined by Equations 5.3 and 5.4, are expressed as linear constraints using a standard method from the Operations Research domain:

$$\Delta_{l,j} - M \cdot (3 - (1 - x_l) - x_j - x_k) \leq \Delta_{l,k}$$
$$\Delta_{j,l} - M \cdot (3 - (1 - x_l) - x_j - x_k) \leq \Delta_{k,l}$$

We denote by M a large positive constant.

In order to find the maximal mergeable set Γ_i^{max} of node v_i, we maximize the sum of x_j variables that satisfy the previous constraints. Then, we come up with the following ILP representation:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} x_j & & \\
\text{subject to} \quad & x_j = 0 & & \text{if } \pi(v_j) \neq \pi(v_i) \\
& \Delta_{l,j} - M \cdot (3 - (1 - x_l) - x_j - x_k) \leq \Delta_{l,k} & & \forall\ j \neq l,\ k \neq l \\
& \Delta_{j,l} - M \cdot (3 - (1 - x_l) - x_j - x_k) \leq \Delta_{k,l} & & \forall\ j \neq l,\ k \neq l
\end{aligned} \tag{5.5}$$

In the previous ILP formulation (Equation 5.5), the number of introduced zero-one variables x_j is equal to n. Regarding the number of constraints, it is bounded from above by $2n^3 + n$. Hence, the proposed ILP formulation has a polynomial size in the number of nodes (i.e. number of sub-tasks in the DAG task to reduce).

Example 5.7. *In this example, we present the result obtained by applying the ILP formulation on the DAG task defined in Figure 5.11 in order to find a maximal mergeable set Γ_1^{max} of the node v_1 (i.e. sub-task $\tau_{1,1}$) that runs on core π_1.*

The dependencies matrix Δ corresponding to the DAG task defined in Figure 5.11 is equal to:

$$\Delta = \begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

The vector of zero-one variables x_j found by an ILP solver [105] that maximize the defined ILP problem (Equation 5.5) is:

$$\{x_1, \ldots, x_9\} = \{1,\ 1,\ 1,\ 1,\ 1,\ 0,\ 0,\ 0,\ 0\}$$

For the nodes that run on a different core than v_1 (i.e. run on π_2), the first constraint in ILP formulation (Equation 5.5) is verified, i.e. $x_7 = 0$ and $x_8 = 0$.

In addition, the solution obtained (the vector of x_j's) satisfies the two other constraints of the ILP formulation (Equation 5.5). For instance, if $j = 1$, $k = 5$ and $l = 8$, we have $x_1 = 1$, $x_5 = 1$, $x_8 = 0$, $\Delta_{1,8} = 1$, $\Delta_{8,1} = 0$, $\Delta_{5,8} = 1$ and $\Delta_{8,5} = 0$. Hence, the following constraints are met :

$$\Delta_{8,1} - M \cdot (3 - (1 - x_8) - x_1 - x_5) \leq \Delta_{8,5}$$

$$\Delta_{1,8} - M \cdot (3 - (1 - x_8) - x_1 - x_5) \leq \Delta_{5,8}$$

When $j = 5$, $k = 1$ and $l = 8$, the following constraints are also met:

$$\Delta_{8,5} - M \cdot (3 - (1 - x_8) - x_5 - x_1) \leq \Delta_{8,1}$$

$$\Delta_{5,8} - M \cdot (3 - (1 - x_8) - x_5 - x_1) \leq \Delta_{1,8}$$

5.3.2 Heuristic based approach

In this part, we provide a graph reduction algorithm based on a greedy heuristic that runs in polynomial time. This heuristic searches for two mergeable nodes that satisfy the three conditions in Definition 5.1 and merges them. It proceeds iteratively until no more pairs of mergeable nodes are found. This approach allows to reduce the computational complexity compared to the ILP based approach (Section 5.3.1). But, it does not guarantee reducing the number of nodes in the DAG task to the minimum.

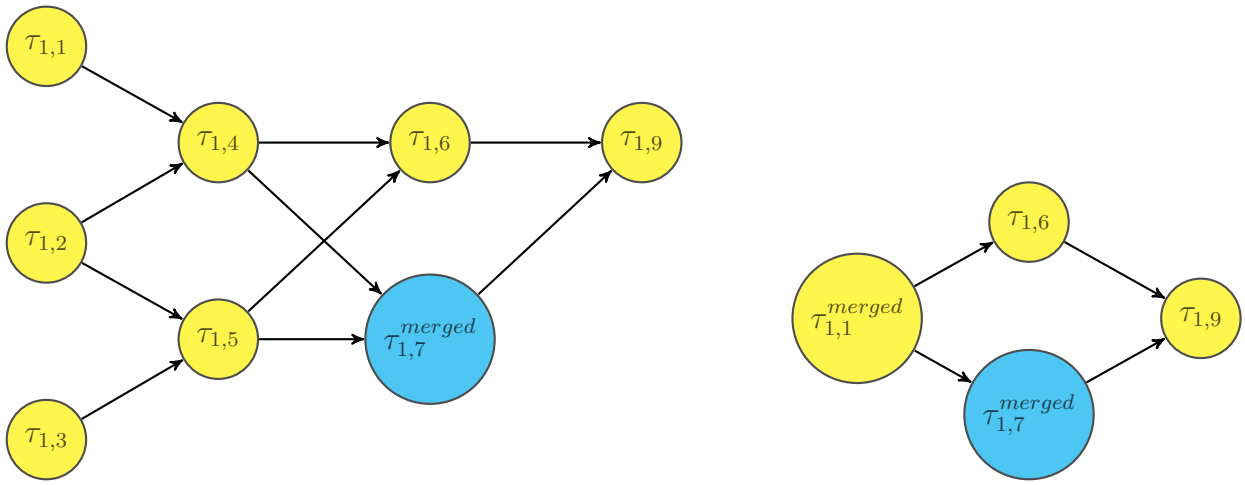
Algorithm 6 describes how this heuristics works. We start by initializing the resulting DAG task τ_i' and a Boolean variable that indicates if any merge operation happened during the last iteration of the *while* loop (lines $3 - 13$). Inside the *while* loop, we use two *for* loops (lines $5 - 6$) to go through the graph nodes and examine if there are any mergeable pairs of sub-tasks $\tau_{i,j}$ and $\tau_{i,k}$ (line 7). If so, we merge them (line 8) and we set the Boolean variable *merged* to *true* in order to make the *while* loop (line 3) recheck if any other pairs of sub-tasks in the new DAG task τ_i' could be merged.

We note that Algorithm 6 has three nested loops that call the function "*merge*". The latter has an $O(n_i)$ time complexity, where n_i is the number of sub-tasks in a DAG task τ_i. Hence, we deduce that our proposed graph reduction algorithm has a polynomial time complexity equal to $O(n_i^4)$.

Algorithm 6: Graph reduction heuristic

Data: τ_i a DAG task and $\pi(\cdot)$ the core mapping
Result: τ_i' the reduced DAG task

```
1  τ_i' = τ_i
2  merged = true
3  while merged do
4  |  merged = false
5  |  for τ_{i,j} ∈ τ_i' do
6  |  |  for τ_{i,k} ∈ τ_i' do
7  |  |  |  if π(τ_{i,j}) = π(τ_{i,k}) and pred(τ_{i,j}) \ {τ_{i,k}} = pred(τ_{i,k}) \ {τ_{i,j}} and
   |  |  |   succ(τ_{i,j}) \ {τ_{i,k}} = succ(τ_{i,k}) \ {τ_{i,j}} then
8  |  |  |  |  τ_i' = merge(τ_{i,j}, τ_{i,k})
9  |  |  |  |  merged = true
10 |  |  |  |  break
11 |  |  |  end
12 |  |  end
13 |  end
14 end
15 return τ_i'
```



(a) Heuristic based approach (Section 5.3.2) **(b)** ILP based approach (Section 5.3.1)

Figure 5.12: Reduction of DAG task defined in Figure 5.11

Example 5.8. *In this example, we apply our graph reduction Algorithm 6 on the DAG task defined in Figure 5.11. In Figure 5.12, we illustrate the reduced DAGs obtained by our heuristic (cf. Figure 5.12a) and by the ILP based approach (cf. Figure 5.12b).*

We note that our reduction heuristic merges the two sub-tasks that execute on core π_2. However, it does not come out with the maximal mergeable set $\Gamma_{1,1}^{max}$ of sub-task $\tau_{1,1}$ that runs on core π_1 as the ILP based approach does.

From Figure 5.12, we note that we should check different combinations of sets that include sub-task $\tau_{1,1}$ in order to find the maximal mergeable set $\Gamma_{1,1}^{max}$. Thus, the problem of determining the maximal mergeable set of a sub-task seems to be an NP-hard problem. In Appendix B, we give some ideas on proving the NP-hardness of this problem.

Remark. *In the DAG defined in Figure 5.11, if there are precedence constraints from $\tau_{1,1}$ to $\tau_{1,2}$ and from $\tau_{1,2}$ to $\tau_{1,3}$, then our graph reduction heuristic will find the optimal solution found by the ILP based approach. In fact, with these additional edges, our heuristic succeed to merge sub-tasks $\tau_{1,1}$ and $\tau_{1,2}$ together. It also merges $\tau_{1,3}$ with $\tau_{1,5}$ then $\tau_{1,4}$ with $\tau_{1,5}^{merged}$. Finally, it merges $\tau_{1,1}^{merged}$ with $\tau_{1,5}^{merged}$. Therefore, we obtain the same reduced graph as in Figure 5.12b.*

As mentioned previously in Section 5.1.2, the priority ordering of sub-tasks transforms the partial order defined by precedence constraints to a total order. Thus, the priority order between two sub-tasks is equivalent to a precedence constraint from the highest priority sub-task to the lowest one. We deduce that the priority ordering of sub-tasks helps our graph reduction heuristic to perform better.

5.4 Integrated Scheduling Methodology

In this section, we present an integrated scheduling methodology that joins several previously proposed scheduling techniques. We apply these techniques together, as described in the workflow below (cf. Figure 5.13), in order to reduce the response time and enhance the schedulability of the system. This workflow takes as input a DAG task set defined by the execution times, period and deadline for each DAG task. After going through the different phases of the workflow, we obtain a partitioned DAG task set with priority defined at the task and sub-task levels. Hence, we apply, on the resulting task set, our partitioned and fixed-priority schedulability analysis (i.e. RTA) proposed in Chapter 3 for deterministic timing parameters and in Chapter 4 for probabilistic ones.

Moreover, we use the result of the schedulability analysis to refine different steps of the workflow. For instance, the priority assignment at the task level based on Audsley's algorithm (Section 5.1.1.2) uses the result of the schedulability analysis to enhance the priority assignment.

The workflow presented in Figure 5.13 is composed of several steps:

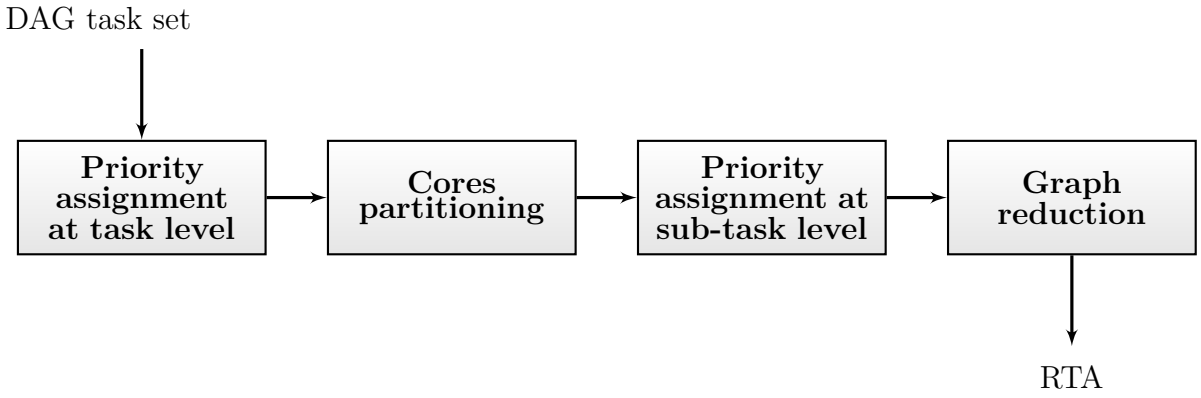


Figure 5.13: Workflow of applying proposed scheduling techniques

1. **Priority assignment at the task level:** In this step, we define priorities for DAG tasks using DM policy or Audsley's algorithm proposed in Section 5.1.1. If the task set is not yet partitioned (i.e. first run of the workflow), we assign priority with DM policy since Audsley's algorithm requires a task set on which we can apply the schedulability test.

2. **Cores partitioning:** In this step, we go through DAG tasks in descending order of their priorities defined in the previous step and we allocate a core for each sub-task as described in Section 5.2.

3. **Priority assignment at the sub-task level:** In order to attribute a priority for each sub-task, we use the defined core partitioning and one of the priority ordering methods provided in Section 5.1.2.

4. **Graph reduction:** Here, we use the defined core partitioning and sub-tasks priority ordering and we apply one of the methods described in Section 5.3. These methods allow us to reduce the size of the DAGs and consequently to decrease the computational complexity of the schedulability test.

6 Evaluation Results

Contents

In this chapter, we present the evaluation of different scheduling techniques and schedulability analysis proposed in the previous Chapters 3, 4 and 5. First, we describe the experimental setup used for the experiments conducted. Second, we evaluate the proposed algorithms on randomly generated DAG task sets. Finally, we apply our proposed probabilistic schedulability analysis on a real use case of a PX4 autopilot[1] programs used for the control of several types of UAVs[2] and mobile robots.

[1] `https://en.wikipedia.org/wiki/PX4_autopilot`
[2] `https://docs.px4.io/master/en/airframes/airframe_reference.html`

6.1 Experimental Setup

In this section, we describe the experimental setup used for all evaluation experiments. We start by presenting how our DAG tasks generator works and we specify the different parameters considered for the generation of graphs and timing parameters of each sub-task. Then, we present the SimSo simulator [114] used for simulating independent real-time tasks on either a single core or a multi-core processor and we explain our implemented extension[3] of this tool.

6.1.1 Random Generation of DAG Tasks

The generation procedure of DAG tasks is composed of several steps. We describe below these steps as well as the parameters involved in each step:

- **Step 1:** We use "`randfixedsum`" algorithm [115] to generate n task utilizations from a given total utilization U_t equal to 70% of the system capacity i.e. $U_t = 0.7 \times m$, where m is the number of cores.
- **Step 2:** We use "log-uniform" distribution to generate tasks' periods in the range $[10, 1000ms]$ and we set the deadlines equal to the periods ($D_i = T_i,\ \forall\ i \in \{1, \dots n\}$).
- **Step 3:** We compute the execution time of each DAG task using this formula $C_i = T_i \times U_i,\ \forall\ i \in \{1, \dots n\}$, where U_i is the individual utilization of the DAG task τ_i.
- **Step 4:** We split the total number of sub-tasks $N_{\text{sub-task}}$ into n numbers that define the number of sub-tasks n_i on each graph τ_i, i.e. $N_{\text{sub-task}} = \sum_i n_i$
- **Step 5:** We generate n_i sub-tasks per DAG task τ_i. We define their execution times $C_{i,j},\ \forall\ j \in \{1, \dots n_i\}$ in such a way to sum up the total execution time of τ_i i.e. $\sum_j C_{i,j} = C_i$. To do so, we use "`UUniFast`" algorithm [116] and we round the results to obtain integer execution times.
- **Step 6:** If we need to generate probabilistic task sets, we build a discrete probability distribution to represent the pWCET of each sub-task $\tau_{i,j}$. This distribution have $K_{\mathcal{C}_{i,j}}$ possible values and its expected value is equal to $C_{i,j}$.
- **Step 7:** We create a graph with n_i sub-tasks for each task τ_i. We generate edges between nodes randomly and we ensure that there is no cycle.

[3] `https://github.com/SlimBenAmor/simso`

- **Step 8:** We assign priority to DAG tasks based on their periods according to the DM policy [13] (equivalent to RM [12] since $D_i = T_i,\ \forall\ i \in \{1, \dots n\}$).
- **Step 9:** We define sub-task allocation randomly or by the mean of heuristics.
- **Step 10:** We define static priority for the sub-tasks according to the topological order or using heuristics.

The generation of periods using "log-uniform" distribution is studied by Davis et al. [117]. They show that it helps to avoid the problem of bias between the magnitude of periods range and the schedulability test efficiency.

The hyperperiod of a task set increases exponentially in the total number of prime factors composing each period. In order to avoid significantly large hyperperiods (i.e. study interval) of the generated task sets, we select the two smallest periods generated with log-uniform distributions and round the other periods to the nearest multiple of the two selected periods.

We deploy layer-by-layer with the same edge probability method [118, 119] to generate graphs with unbiased structures. In fact, we group sub-tasks into layers. We use a random number of layers and a random size for each layers. Then, we connect arbitrary sub-tasks of a given layer to sub-tasks in subsequent layers. The probability of edge creation between two sub-tasks is equal to $p = 0.2$. This technique allows to avoid cycles formation but it may cause a disconnected graph. Therefore, we check the connectivity at end of the process and we create additional edges if needed. In appendix C, we give examples of randomly generated DAG tasks with the procedure described above.

The random generation of DAG tasks use several parameters. Below, we enumerate the different input parameters for the generation procedure:

- n: the number of DAG tasks.
- m: the number of cores.
- U_t: the total utilization of the task set.
- $N_{\text{sub}-\text{task}}$: the total number of sub-tasks in the whole task set.
- p_{edge}: the probability of generating an edge between two sub-tasks belonging subsequent levels.
- $K_{\mathcal{C}}$ the number of possible values in each distribution in case of probabilistic task set.

In the experiences presented below, we run each algorithm on 100 task sets and we compare the average performance over these 100 task sets. When not mentioned below, each generated task set is composed of $N = 5$ DAG tasks and $N_{\text{sub-task}} = 100$ sub-tasks scheduled on $m = 4$ cores with a total utilization $U_t = 2.8$. We limit the number of tasks and sub-tasks because of the computational complexity of some algorithms from literature used for comparison (e.g. MILP approach [55] for RTA).

6.1.2 SimSo Simulator

SimSo is a simulation tool developed by Chéramy et al. [114] to evaluate real-time scheduling algorithms. It supports single and multi-core processor scheduling. It also supports several models and scheduling policies. However, it does not deal with DAG task models, priority definition at the sub-task level and probabilistic execution times. Therefore, we adapted the source code[4] to be able to add precedence constraints inside tasks and to specify a static sub-task-to-core allocation. We also define fixed-priority at the task and sub-task levels. Hence, we could simulate the generated task sets and derive their response times from the simulator events log.

Since we use a partitioned scheduling on identical multi-core processors, our problem could be seen as several single core processor problems once the allocation is done [14]. On the other hand, Baruah and Burns [19] show that fixed-priority and preemptive scheduling with release jitter on a single core processor is sustainable with respect to the period. Moreover, our priority assignment algorithm at the sub-task level is consistent with the precedence constraints and respects the topological order, i.e. a predecessor sub-task always have a higher priority than its successors. Thus, the SimSo simulation of a DAG task model with precedence constraints is also sustainable. Hence, if a sporadic DAG task model is schedulable with the minimum inter-arrival time as the period for all DAG tasks, then it remains schedulable with higher periods and sporadic arrivals.

We deduce that if a periodic DAG task model is schedulable on a partitioned multi-core processor, then the sporadic task set is also schedulable. Thus, it is sufficient to study the periodic system on the feasibility interval [18] that equals to the hyperperiod (i.e. least common multiple of the periods) because the same scheduling will be repeated after each hyperperiod. In fact, there is no backlog (unfinished jobs) passed from a hyperperiod to the next one because unfinished jobs are dropped at the end of their deadlines and we assume that the deadline is less than or equal to the period for each task. Consequently, the maximum response time observed during the simulation of a generated task set over a hyperperiod, is equal to the WCRT.

[4] `https://github.com/SlimBenAmor/simso`

6.2 Evaluation of RTA and Scheduling Techniques

6.2.1 Response Time Analysis

6.2.1.1 Deterministic approach

First experiment

First, we apply our RTA proposed in Section 3.2.3.1 on a DAG task model with deterministic execution times and priority defined at the task level only using DM algorithm [13]. Then, we compare our method and the holistic approach [1] to the MILP based approach proposed by Fonseca et al. [55] by computing the WCRT ratio over the one obtained by MILP approach.

Table 6.1: WCRT ratio regarding the MILP based approach [55]

	Min ratio	**Avg ratio**	**Max ratio**
Our RTA (sec. 3.2.3.2)	1	1.61	8
Holistic [1] (sec. 3.2.2.1)	1	2.52	9.73
MILP [55]	1	1	1

Table 6.1 shows that our RTA analysis and the holistic one overestimate the WCRT computed by the MILP based approach [55]. However, our approach introduces less over-estimation and pessimism than the holistic approach. On average, our estimated WCRT is 1.61 times larger than the WCRT of MILP approach, while the WCRT of the holistic approach is 2.52 times larger. Thus, the WCRT of the holistic approach is about 1.5 times larger, on average, than our WCRT. Moreover, for some generated task sets, we have large difference between MILP approach and other approaches. The computed WCRT reaches even up to 8 and 9 times larger (Max ratio column in Table 6.1). Meanwhile, for other task sets, we obtain exactly the same WCRT for the three approaches (Min ratio column in Table 6.1).

Table 6.2: Comparison of run-time of RTA

	Min run-time	**Avg run-time**	**Max run-time**
Our RTA (sec. 3.2.3.2)	0.001 s	2.1 s	11.2 s
Holistic [1] (sec. 3.2.2.1)	0.0009 s	1.9 s	10.17 s
MILP [55]	0.01 s	74.7 s	7648 s

Table 6.2 illustrates the run-time performance of the three RTA approaches. We note that run-times of our RTA and holistic analysis are comparable and they are much faster than MILP approach. On average, our algorithm takes 2.1 seconds

to deliver a WCRT estimation and the holistic analysis takes 1.9 seconds, while MILP based analysis takes 74.7 seconds. Besides, the maximum run-time, over 100 generated task sets, is 11.2 seconds for our approach and 10.17 seconds for the holistic approach, while MILP analysis takes more than 2 hours. Consequently, it is clear that the MILP-based schedulability test could not be deployed in an online scheduler of an interactive real-time system due to its high complexity and run-time overhead. However, it could be used offline to generate scheduling tables or as schedulability analysis for real-time systems design.

Second experiment

In this experiment, we evaluate the 5 RTA methods proposed in Chapter 3. We start by computing the response time of 100 randomly generated task sets composed each of $n = 10$ DAG tasks and a total number of sub-tasks $N_{sub-task} = 100$ executed on $m = 2$ cores that have a utilization equal to 50%. We define priority at the task and sub-task levels using DM algorithm [13] and our sub-task priority assignment heuristic (Section 5.1.2.3) respectively. Then, we compute the ratios of the response times obtained with each RTA method over the exact one obtained with SimSo simulation for a hyperperiod.

Table 6.3: WCRT ratios regarding the SimSo simulation and run-times of the 5 RTA methods proposed in Chapter 3.

	Min ratio	**Avg ratio**	**Max ratio**	**Min run-time**	**Avg run-time**	**Max run-time**
RTA in Sec. 3.2.2.1	1	3.01	9.35	0.121 s	**0.183** s	0.62 s
RTA in Sec. 3.2.2.2	1	2.9	8.51	0.126 s	0.189 s	0.629 s
RTA in Sec. 3.2.2.3	1	2.79	8.34	0.127 s	0.19 s	0.631 s
RTA in Sec. 3.2.3.1	1	1.66	15.41	**0.111** s	0.191 s	**0.619** s
RTA in Sec. 3.2.3.2	1	**1.2**	**3.73**	0.191 s	0.298 s	0.822 s
RTA SimSo	1	1	1	0.54 s	54.96 s	170.57 s

From Table 6.3, we note that, on average, the two RTA methods based on our characterization of the worst-case arrival patterns (in Sections 3.2.3.1 and 3.2.3.2) reduce the over-estimation of the response time compared to other methods based on the arrival patterns described in [1]. However, for some generated task set our first characterization of the worst-case arrival patterns (in Sections 3.2.3.1) could be very pessimistic. It over-estimate the actual response time about 15 and half times ((Max ratio column in Table 6.3).

Regarding the run-times of the different RTA methods proposed, we note that they have similar run-times with small variations between 0.1 s and 0.85 s. Indeed,

these run-times are less than 1 second since RTA methods proposed in Chapter 3 are based on iterative equations with a polynomial complexity.

Table 6.4: Number of DAG task that have strictly greater response time using one RTA method (from rows) compared to the other (from columns)

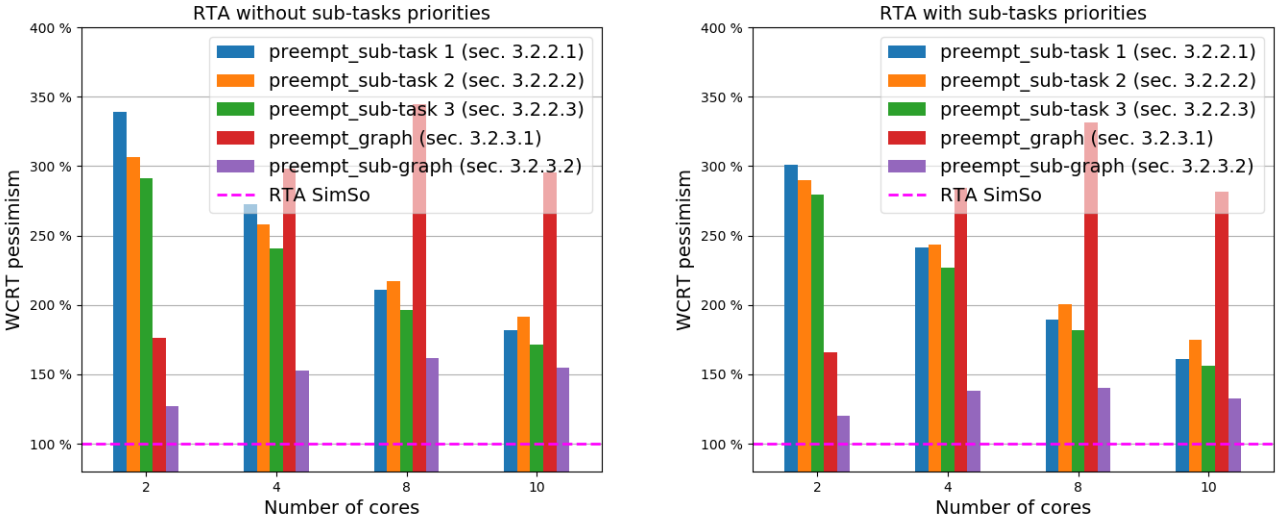
	RTA in Sec. 3.2.2.1	**RTA in Sec. 3.2.2.2**	**RTA in Sec. 3.2.2.3**	**RTA in Sec. 3.2.3.1**	**RTA in Sec. 3.2.3.2**
RTA in Sec. 3.2.2.1	0	556	722	787	891
RTA in Sec. 3.2.2.2	221	0	746	800	908
RTA in Sec. 3.2.2.3	0	0	0	732	842
RTA in Sec. 3.2.3.1	107	108	114	0	258
RTA in Sec. 3.2.3.2	1	1	1	13	0

Even if the WCRT ratio of some RTA methods may seems comparable in Table 6.3, these methods are not actually comparable. In fact, for some generated task sets, RTA methods based on our characterization of the worst-case arrival patterns (in Sections 3.2.3.1 and 3.2.3.2) reduce the over-estimation of the response time, while for other task sets it is the inverse. In Table 6.4, we illustrates the number of generated DAG tasks that have a response time with the RTA method in a given row strictly greater than the RTA method in a given columns (i.e. $R(row) > R(column)$). We recall that there is a total number of DAG tasks equal to 1000 (i.e. 10 DAG tasks $\times$100 task sets). The number of DAG tasks in a diagonal cell in Table 6.4 is equal to 0 because using the same RTA methods for a DAG task always results in equal response times (and not strictly greater).

We note that the third RTA method based on the worst-case arrival patterns described in [1] (third row in Table 6.4), always yields a response time less than or equal to the one computed by the two other RTA methods based on the same arrival patterns. We also note that the second RTA method based on our characterization of worst-case arrivals (fifth row in Table 6.4) provides almost the lowest over-estimation of the response time except for 14 generated DAG tasks over 1000. Nonetheless, other RTA methods provide responses times that are sometimes lower and sometimes higher when compared to each other. Hence, these methods are not comparable between each other.

Third experiment

In this experiment, we apply the 5 RTA methods proposed in Chapter 3 on 100 task sets generated with different configurations and generation parameters. Indeed, we vary the number of cores m among 2, 4, 8 and 10 with a constant utilization equal to 50% of the capacity of the system. The number of DAG task n could be equal to 6, 8, 10 or 12. The total number of sub-tasks $N_{sub-task}$ varies among 40, 60, 80 and 100. We also define priority at the task level only for one set of experiments and at both levels for another set of experiments. Then, we compute and compare the ratios of the response times obtained with each RTA method over the exact one obtained with SimSo simulation.



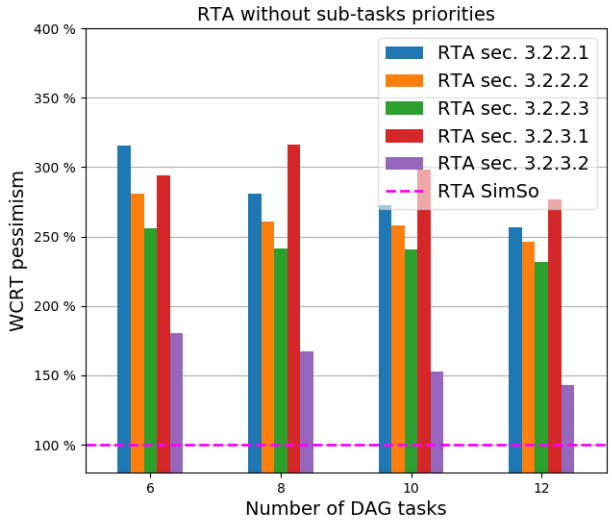
(a) Average WCRT ratio with priority defined at the task level only.

(b) Average WCRT ratio with priority defined at the task and sub-task levels.

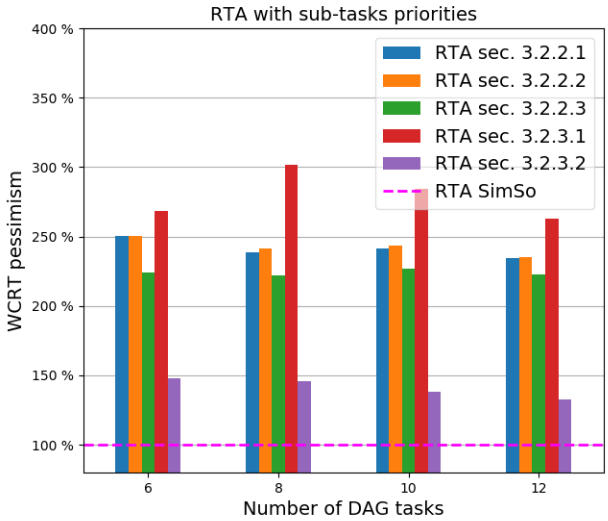
Figure 6.1: Average WCRT ratio of 100 generated task sets that have $n = 10$ DAG tasks and $N_{sub-task} = 100$ sub-tasks while varying the number of cores.

From the different figures above, we note that the response time is reduce for all RTA methods when we define priority at the sub-task level using our assignment heuristic (Section 5.1.2.3).

In Figure 6.1, we note that, on average, the response times of the RTA methods based on the worst-case arrival patterns described in [1] (Sections 3.2.2.1, 3.2.2.2 and 3.2.2.3), are reduced when the number of cores increases. This is because the number of preempting sub-tasks from higher priority DAGs that are executed on the same core become smaller since the total number of sub-tasks ($N_{sub-task} = 100$) is partitioned on more cores. However, the response times of other RTA methods increase with the number of cores. Indeed, the performance of these methods

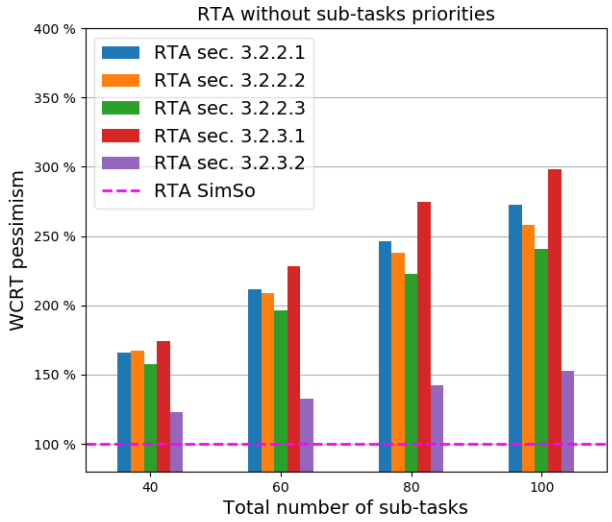


(a) Average WCRT ratio with priority defined at the task level only.

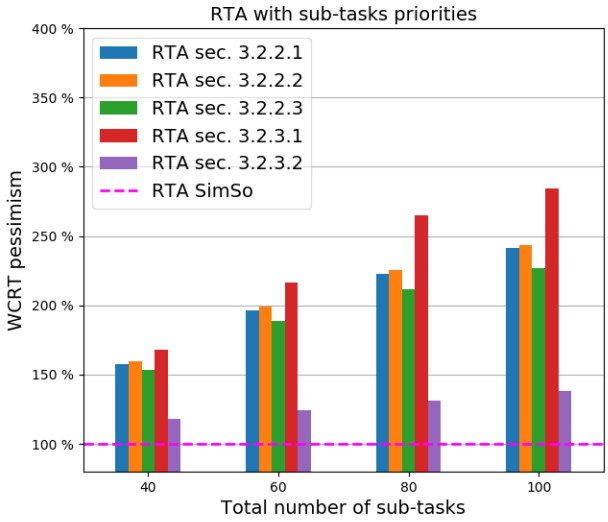


(b) Average WCRT ratio with priority defined at the task and sub-task levels.

Figure 6.2: Average WCRT ratio of 100 generated task sets that have $N_{sub-task} = 100$ sub-tasks executed on $m = 4$ cores while varying the number of DAG tasks.



(a) Average WCRT ratio with priority defined at the task level only.



(b) Average WCRT ratio with priority defined at the task and sub-task levels.

Figure 6.3: Average WCRT ratio of 100 generated task sets that have $n = 10$ DAG tasks executed on $m = 4$ cores while varying the number of sub-tasks.

depends a lot on the partitioning of sub-tasks and when the number of cores increases, the partitioning problem become harder.

In Figure 6.3, we note that the response times for all RTA methods increase with

the total number of sub-tasks $N_{sub-task}$ because the communication and interference between different sub-tasks and cores increase. In Figure 6.2, we also note that the response times slightly decrease when the number of DAG tasks n increases. This is because the number of sub-tasks in each DAG task become smaller since the total number of sub-tasks is fixed $N_{sub-task} = 100$.

6.2.1.2 Probabilistic approach

Response time equations based on probabilistic operators

We evaluate our probabilistic response time equations presented in Section 4.2.2 by comparing them to the deterministic approach. The deterministic analysis is based on worst case reasoning. Hence, it considers the highest execution times in the pWCET distributions and it declares a task set schedulable when the probabilistic analysis finds a probability of schedulability equals to 100%. We note that, in Figure 6.4, none of the generated task sets reaches the probability of 100% so they won't be evaluated as schedulable using the deterministic analysis. However, about half of generated tasks are schedulable with high probability (more than 80%) which highlights the pessimism of the worst case reasoning of the deterministic analyses. There is a significant number of task sets with 0% probability to be schedulable (not schedulable under any timing parameters values). This is explained by the random generation of timing and precedence constraints that may be too stringent to be respected.

Bayesian network

In this experiment, we apply the Bayesian network inference on 10 generated task sets each composed of a single DAG task with probabilistic execution times. Each DAG task contains $N_{sub-task}$ sub-tasks that are executed on $m = 2$ cores. The execution time distribution of each sub-task have $K_{\mathcal{C}}$ possible values. We vary $N_{sub-task}$ among 5, 6 and 7. We also vary $K_{\mathcal{C}}$ among 3, 4 and 5.

First, we compute the cumulative distribution of the response time of the 10 DAG tasks generated using four different methods: the exact method that explores *all combinations*, the exact Bayesian inference with *Variable Elimination*, the approximate Bayesian inference with *sampling* and the response time equations with *probabilistic operators* (Section 4.2.2). In our case, the response time distribution of the DAG task have on average about 110 points. Second, we calculate the maximum difference over all points between each method and the exact one that explores *all combinations* and we compare the average of this maximum difference over the 10 DAG tasks generated. We limit the value of some parameters such as the number

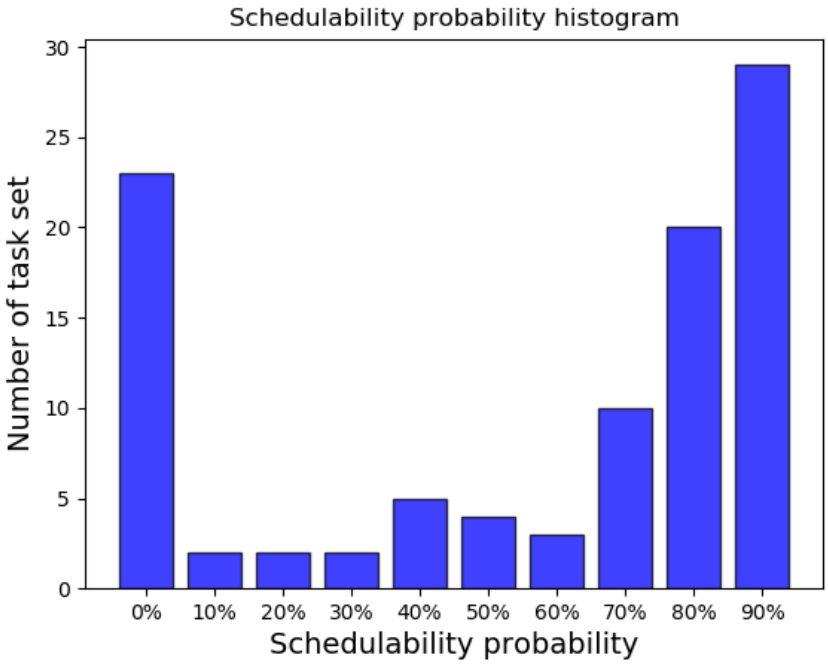


Figure 6.4: Histogram of schedulability probability for 100 task sets randomly generated

of DAG task generated, the number of sub-tasks $N_{sub-task}$ and the size of pWCET distributions $K_{\mathcal{C}}$ due to the high computation complexity of some methods used (i.e. *all combinations* and *Variable Elimination*) that increases exponentially.

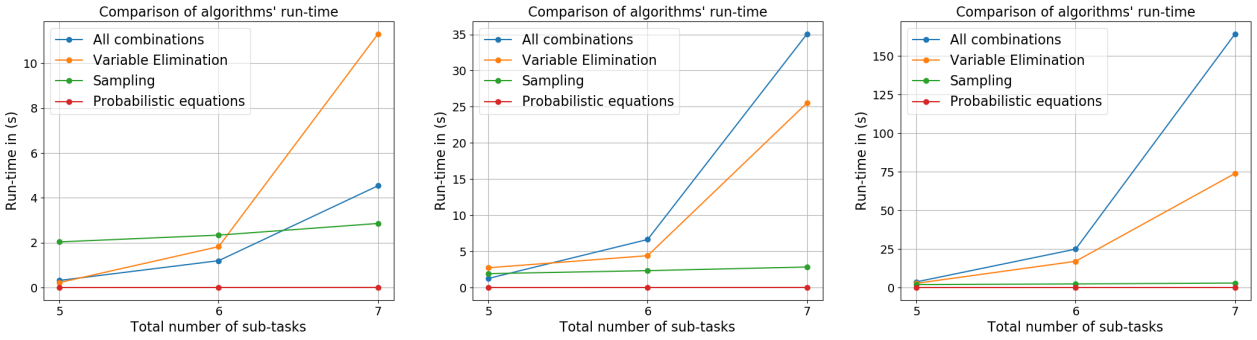
Table 6.5: Maximum difference of CDF compared to *all combinations* method for DAG tasks with $N_{sub-task} = 6$ sub-tasks and different size of pWCET distributions $K_{\mathcal{C}}$.

$\mathbf{K_{\mathcal{C}}}$	**3**	**4**	**5**
Variable Elimination	9.44×10^{-17}	2.22×10^{-16}	2.22×10^{-16}
Sampling	1.99×10^{-3}	1.14×10^{-3}	1.19×10^{-3}
Probabilistic operators	1.88×10^{-2}	1.59×10^{-2}	2.23×10^{-2}

Table 6.6: Maximum difference of CDF compared to *all combinations* method for DAG tasks with a size of distributions $K_{\mathcal{C}} = 5$ and different number of sub-tasks $N_{sub-task}$.

$\mathbf{N_{sub-task}}$	**5**	**6**	**7**
Variable Elimination	1.89×10^{-16}	2.22×10^{-16}	5.32×10^{-16}
Sampling	$9.48. \times 10^{-4}$	1.19×10^{-3}	9.18×10^{-4}
Probabilistic operators	1.68×10^{-2}	2.23×10^{-2}	1.4×10^{-2}

From Tables 6.5 and 6.6, we note that the maximum difference of CDF between the *Variable Elimination* and *all combinations* methods is almost equal to 0 (about 10^{-16}). Thus, we deduce that the *Variable Elimination* method always provides the same response time distribution as the exact method (i.e. *all combinations*). However, *Sampling* and *Probabilistic operators* methods have a remarkable difference (between 10^{-3} and 10^{-2}). For the *Sampling* method, this difference is inversely proportional to the number of samples $N_s = 1000$.



(a) Average of run-times for a size of distribution $K_{\mathcal{C}} = 3$

(b) Average of run-times for a size of distribution $K_{\mathcal{C}} = 4$

(c) Average of run-times for a size of distribution $K_{\mathcal{C}} = 5$

Figure 6.5: Average of run-times of different probabilistic RTA methods for several numbers of sub-tasks $N_{sub-task}$ and sizes of distributions $K_{\mathcal{C}}$.

From Figure 6.5, we note that the run-times of different algorithms increase with the number of sub-tasks $N_{sub-task}$ and the size of distributions $K_{\mathcal{C}}$. The run-times of *Variable Elimination* and *all combinations* methods increase exponentially. For a size of distributions $K_{\mathcal{C}} = 3$, *Variable Elimination* method have higher run-time than *all combinations*. However, the run-time of *all combinations* method increases faster for $K_{\mathcal{C}} = 4$ and $K_{\mathcal{C}} = 5$.

On the other hand, the *Sampling* and *Probabilistic operators* methods have a polynomial complexity. Hence, their run-times barely increase for small variation of number of sub-tasks $N_{sub-task} \in \{5, 6, 7\}$ and size of distributions $K_{\mathcal{C}} \in \{3, 4, 5\}$.

C-space and SVM classifier

In this experiment, we generate 10 task sets each composed of a single DAG task with probabilistic execution times. Each DAG task contains $N_{sub-task} \in \{5, 6, 7, 8\}$ sub-tasks that are executed on $m \in \{1, 2, 4, 8\}$ cores. The execution time distribution of each sub-task have $K_{\mathcal{C}} = 5$ possible values. Then, we compute the DMPs of each DAG task using *all combinations* and *SVM* methods (described in Section 4.4).

SVM classifier with linear Kernel First, we start by evaluating the *SVM* classifier with a linear kernel. Therefore, we use DAG tasks that are executed on a single core processor ($m = 1$) and we compute the difference between DMPs obtained with *linear kernel SVM* and *all combinations* methods (i.e. DMP(SVM)−DMP(all combinations)).

Table 6.7, illustrates the minimum, average and maximum of the DMP differences over the 10 generated DAG tasks with a size of distributions $K_{\mathcal{C}} = 5$ and different numbers of sub-tasks $N_{sub-task}$. We note that the DMP difference is always equal to 0 for $N_{sub-task} = 5$ and it is equal to zero 9 times over 10 for other value of $N_{sub-task}$ (because the average DMP difference equal to the maximum DMP difference divided by 10, which is the number of generated DAG tasks). We deduce that the *linear kernel SVM* almost succeed to determine the exact border between the schedulable and non-schedulable regions. The introduced difference for some generated DAG tasks could be caused by a calculation error when normalizing and sampling.

Table 6.7: DMP difference between *linear kernel SVM* and *all combinations* methods for a single core processor.

$\mathbf{N_{sub-task}}$	**5**	**6**	**7**	**8**
Min DMP difference	0	0	0	0
Avg DMP difference	0	2×10^{-8}	6×10^{-8}	2.55×10^{-4}
Max DMP difference	0	2×10^{-7}	6×10^{-7}	2.55×10^{-3}

Table 6.8, shows the result of applying a *linear kernel SVM* on DAG tasks with a size of distribution $K_{\mathcal{C}} = 5$ and $N_{sub-task} = 6$ sub-tasks that are executed on a multi-core processor ($m \in \{2, 4, 8\}$). We note that the minimum DMP difference is not always equal to 0. Hence, SVM classification with linear kernel do not succeed to determine the exact border between the schedulable and non-schedulable regions. Moreover, for some generated DAG tasks the DMP difference between *linear kernel SVM* and *all combinations* methods is negative, which means that computed DMP with *linear kernel SVM* method under-estimates the exact DMP. This is equivalent to an over-estimation of the schedulability probability, which is optimistic and unsafe.

Table 6.8: DMP difference between *linear kernel SVM* and *all combinations* methods for a multi-core processor.

m	**2**	**4**	**8**
Min DMP difference	-6.99×10^{-2}	-1.32×10^{-4}	-5.54×10^{-2}
Avg DMP difference	-1.98×10^{-2}	-7.2×10^{-4}	-1.09×10^{-2}
Max DMP difference	0	4.86×10^{-4}	7.6×10^{-6}

SVM classifier with Gaussian Kernel Second, we evaluate the *SVM* method with a Gaussian kernel for DAG tasks that are executed on a multi-core processor ($m \in \{2, 4, 8\}$). In Table 6.9, we use *Gaussian kernel SVM* method and we compute the minimum, average and maximum of the DMP difference. In Table 6.10, we use *shifted Gaussian kernel SVM* method that consists in shifting the border between schedulable and non-schedulable regions toward the schedulable region. Actually, we shift this border by a half of the standard deviation on each dimension (i.e. $C_i - \frac{\sigma_i}{2}$).

Table 6.9: DMP difference between *Gaussian kernel SVM* and *all combinations* methods for a multi-core processor.

m	**2**	**4**	**8**
Min DMP difference	0	-1.28×10^{-3}	-6.36×10^{-2}
Avg DMP difference	1.05×10^{-2}	4.46×10^{-4}	-1.66×10^{-3}
Max DMP difference	5.12×10^{-2}	3.6×10^{-3}	4.25×10^{-2}

In Table 6.9, there are some negative DMP differences, which means unsafe estimations (under-estimations) of the exact DMPs obtained with *all combinations* method. On the other hand, the minimum DMP difference in Table 6.10 are positive. Thus, there are no negative values obtained with the *shifted Gaussian kernel SVM* method in this experiment. We deduce that this method allows to avoid under-estimation. However, it introduces some pessimism since the maximum DMP differences increase compared to Table 6.9.

Table 6.10: DMP difference between *shifted Gaussian kernel SVM* and *all combinations* methods for a multi-core processor.

m	**2**	**4**	**8**
Min DMP difference	2.14×10^{-4}	0	0
Avg DMP difference	2.64×10^{-2}	9.77×10^{-3}	6.63×10^{-3}
Max DMP difference	8.63×10^{-2}	5.23×10^{-2}	5.09×10^{-2}

Furthermore, we compare the *Gaussian kernel SVM* and *shifted Gaussian kernel SVM* methods using confusion matrices [97]. In fact, we apply these methods on 10 DAG tasks with a size of distribution $K_{\mathcal{C}} = 5$ and $N_{sub-task} = 6$ sub-tasks that are executed on $m = 4$ cores. Then, we compute the average confusion matrix over the 10 generated DAG tasks.

A confusion matrix consists in dividing the set of points in the C-space into four groups regarding if they are predicted correctly with the SVM classifier:

- **True Positive:** schedulable points that are predicted as schedulable.

- **False Positive:** non-schedulable points that are predicted as schedulable.
- **True Negative:** non-schedulable points that are predicted as non-schedulable.
- **False Negative:** schedulable points that are predicted as non-schedulable.

In our case, the total number of points in the C-space (the number of all combinations) is $K_{\mathcal{C}}^{N_{sub-task}} = 5^6 = 15625$

Table 6.11: Average confusion matrix of the *Gaussian kernel SVM* method.

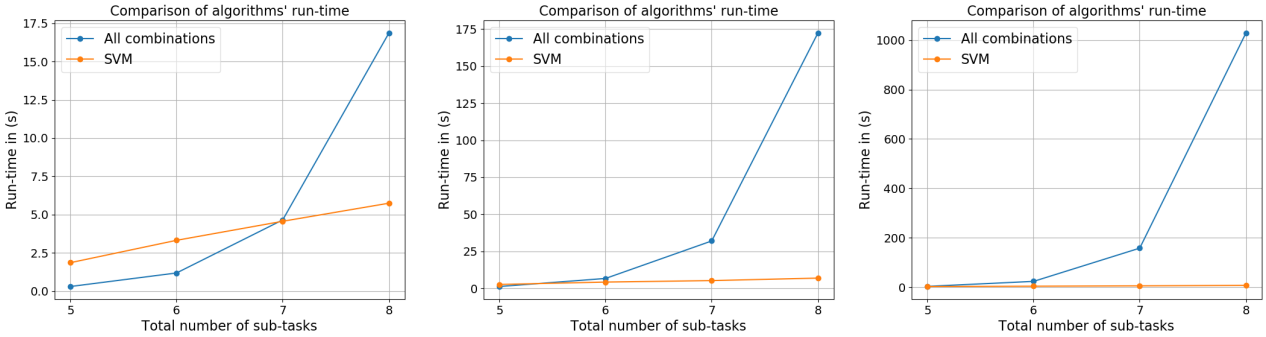
	Actually schedulable	**Actually non-schedulable**
Predicted schedulable	517	11
Predicted non-schedulable	10	15087

Table 6.12: Average confusion matrix of the *shifted Gaussian kernel SVM* method.

	Actually schedulable	**Actually non-schedulable**
Predicted schedulable	475	4
Predicted non-schedulable	52	15094

From Table 6.11 and 6.12, we note that the number of "false positive" points is reduced with the *shifted Gaussian kernel SVM* method from 11 to 4. This allows to avoid an under-estimation of the DMP (i.e. an over-estimation of the schedulability probability). However, the number of "false negative" is increased from 10 to 52, which increases the over-estimation of the DMP and introduces some pessimism. The confusion matrix helps to study the trade-off between safety and pessimism when using *shifted Gaussian kernel SVM* method and to tune the shift value on each dimension in the C-space.

Run-time performance From Figure 6.6, we note that the run-times of *all combinations* and *SVM* methods increase with the number of sub-tasks $N_{sub-task}$ and the size of distributions $K_{\mathcal{C}}$. The run-time of *SVM* method increases linearly since it has a polynomial complexity. However, the run-time of *all combinations* method increases faster due to its exponential complexity.



(a) Average of run-times for a size of distribution $K_{\mathcal{C}} = 3$

(b) Average of run-times for a size of distribution $K_{\mathcal{C}} = 4$

(c) Average of run-times for a size of distribution $K_{\mathcal{C}} = 5$

Figure 6.6: Average of run-times of *all combinations* and *SVM* methods for several numbers of sub-tasks $N_{sub-task}$ and sizes of distributions $K_{\mathcal{C}}$.

Table 6.13: Comparison of different priority assignment algorithm for sub-tasks

	HLFET	**SCFET**	**CPMISF**	**GA**
RTA in Sec. 3.2.3.2	114.66%	119.09%	110.81%	113.9%
Simso simulation	106.57%	110.71%	104.23%	107.18%

6.2.2 Priority Assignment for Sub-tasks

In this section, we evaluate the performance of different priority assignment algorithms at the sub-task level. We compute the response times corresponding to each of this algorithms; Our proposed priority assignment heuristic, HLFET (Highest Levels First with Estimated Times), SCEFT (Smallest Co-levels First with Estimated Times), CPMISF (Critical Path/Most Immediate Successors First) [120, 121] and our proposed genetic algorithm. The response time is computed using two methods: (i) our RTA proposed in Section 3.2.3.1 and (ii) our extension of the SimSo simulator. After that, we calculate the ratio of the obtained response times over the one obtained by our heuristic and we compare them.

Table 6.13 shows that different priority assignment heuristic and genetic algorithm produce, on average, response times larger than the one obtained by our heuristic (ratio $>$ 100%). Besides, we note that, on average, our proposed GA performs better than HLFET and SCEFT heuristics for computed response time with our RTA. However, for response time obtained with SimSo simulator, it perfoms better than SCEFT heuristic only.

6.2.3 Partitioning Heuristic

Figure 6.7 compares the performance of our proposed partitioning heuristic to *Best Fit* and *First Fit* heuristics. We use Simso [114] to simulate the execution of the 100

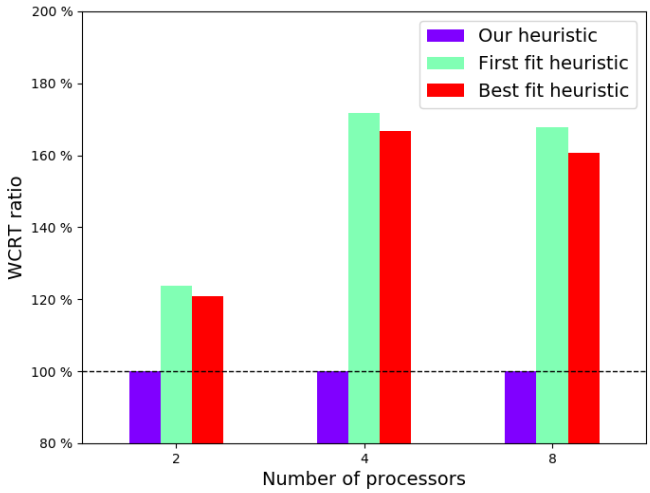


Figure 6.7: WCRT ratio improvement under different partitioning heuristics

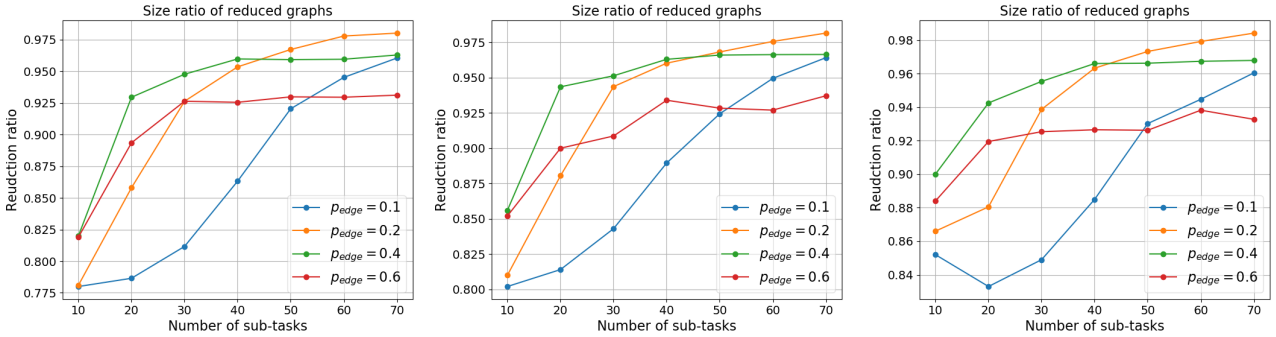
generated task sets on 2, 4 and 8 cores. Then, we determine response time of each DAG task and we compute the ratio of between the response times obtained by *Best Fit* and *First Fit* heuristics over the one obtained by our partitioning heuristic.

We note that our heuristic reduces the response time by mapping sub-tasks in way to increase the possible parallelism. We also note that the difference between our proposed heuristic and others increases when we move from 2 to 4 cores. This is explained by the augmentation of computation resources and the possible parallelism, which confirms that our heuristic exploits the existing parallelism to reduce response time contrary to the two other heuristics that do not consider the structure of DAG and possible parallelism. However, there is almost no difference between using 4 and 8 cores because the maximum parallelism is reached and no further improvement could be made beyond this limit.

6.2.4 Graph Reduction

In order to evaluate the performance of our graph reduction heuristic, we apply it on 100 generated task sets each composed of a single DAG task with different number of sub-tasks $N_{sub-task} \in \{10, 20, \ldots, 70\}$ that are executed on $m \in \{2, 4, 8\}$ cores. The edges between these sub-tasks are generated with a probability $p_{edge} \in \{0.1, 0.2, 0.4, 0.6\}$. Then, we compute the ratio of the size of the reduced DAG over the size of the original one (i.e. the number of sub-tasks $N_{sub-task}$).

From Figure 6.8, we note that the number of cores have not a significant influence on the reduction ratio and the performance of our graph reduction heuristic.



(a) $m = 2$ cores **(b)** $m = 4$ cores **(c)** $m = 8$ cores

Figure 6.8: Average of reduction ratios over 100 DAG tasks randomly generated with different parameters' configurations (p_{edge}, $N_{sub-task}$ and m).

However, the reduction ratio increases with the number of sub-tasks because it becomes harder for our heuristic to find mergeable sub-tasks in larger graphs. We also note that the probability of creating edges p_{edge} has an influence on the reduction ratio. If this probability is not very high (e.g. $p_{edge} = 0.1$, blue lines in Figure 6.8), then there are not many edges and constraints that prevent merging sub-tasks together. In such a case, our heuristic reduces the size of the original graph up to 80% for not very large graphs ($N_{sub-task} \leq 30$). On the other hand, if the probability of creating edges is relatively important (e.g. $p_{edge} = 0.6$, red lines in Figure 6.8), then there are many edges and the order defined by these edges is almost a total order. In Section 5.3.2, we mentioned that defining a total order between sub-tasks helps our graph reduction heuristic to perform better. Thus, our heuristic reduces more (i.e. lower reduction ratio) large graphs ($N_{sub-task} \geq 30$) with an important probability of edge (e.g. $p_{edge} = 0.6$) than other graphs with a medium probability (e.g. $p_{edge} = 0.2$ and $p_{edge} = 0.4$).

From Figure 6.9, we note that the run-time of our graph reduction heuristic increases with the number of sub-tasks $N_{sub-task}$ and with the probability of creating edges p_{edge}. This, is due to the polynomial complexity of our algorithm.

6.3 Use Case: PX4 Autopilot

In this section, we present numerical results obtained for DAG tasks corresponding to the open source PX4 autopilot programs[5] of a drone. The structure of the DAG tasks is illustrated in Figure 6.10.

[5] `https://en.wikipedia.org/wiki/PX4_autopilot`

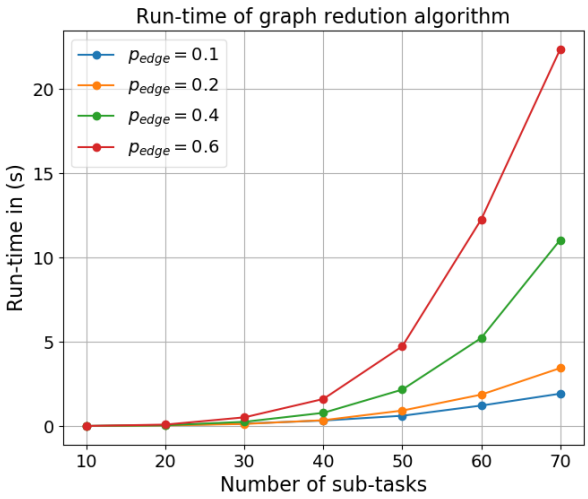


Figure 6.9: Average of run-times over 100 DAG tasks randomly generated with different parameters' configurations (p_{edge}, $N_{sub-task}$) and executed on $m = 4$ cores.

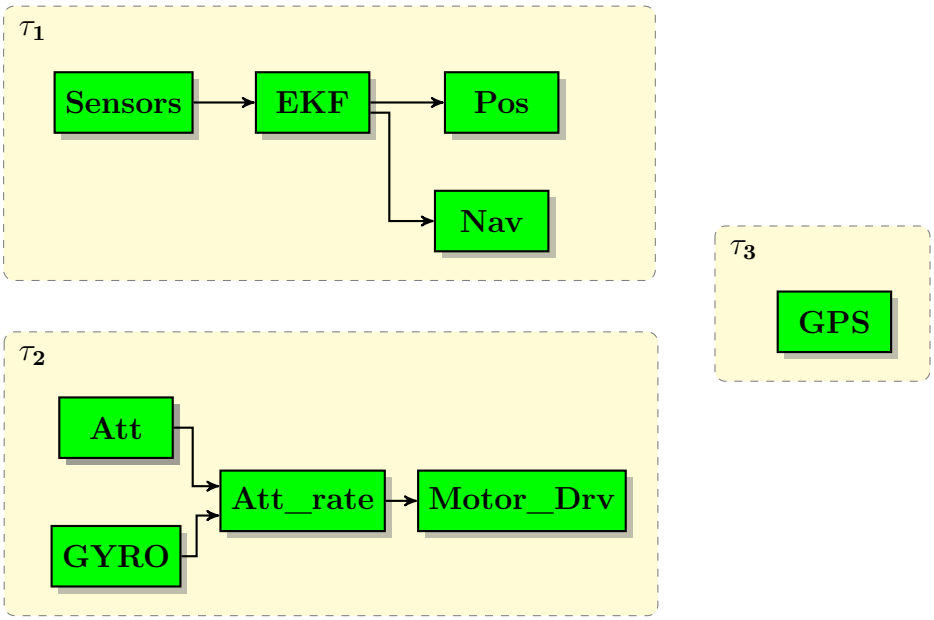


Figure 6.10: DAGs describing the precedence constraints between the sub-tasks of the three tasks representing the PX4 Autopilot programs.

The execution time traces have been obtained from hardware-in-the-loop measurements while the sensors and the output drivers are simulated on predefined flying missions on a Pixhawk 4 hardware[6] on top of a NuttX OS[7]. Moreover, when measured, each sub-task was executed with a highest priority in order to avoid any preemptions from other sub-tasks. The execution time measurements of the

[6] `https://pixhawk.org`
[7] `http://nuttx.org`

sub-tasks are obtained by executing them on a single core processor (ARM family). In order to obtain the probabilistic bounds, we extracted from each empirical distribution several quantiles.

6.3.1 Single Core Processor

Table 6.14: Comparison of computed DMP and drone behavior

Periods	Drone behavior	DMP
3 ms	Could not fly	0.9999
3.5 ms	Could not fly	0.994
4 ms	Poor stability	0.2696
4.5 ms	Medium stability	0.0049
5 ms	Good stability	1.4959×10^{-14}

First, we compute various DMP by setting the period of the PX4 drone autopilot with different values. Then, we compare them to the drone behavior already evaluated with different period values. Results are illustrated in Table 6.14. We note that the obtained DMPs are coherent with the drone behavior obtained from simulation. For instance, when the tasks' period is relatively small, the DMP is very high (near to one) and the drone could not fly because the execution frequency of programs is very high and they cannot finish their execution before the deadline ($D_i = T_i$). On the other hand, DMP is reduced to 10^{-14} when the period is not too small and the drone shows a good stability.

6.3.2 Multi-core Processor

Table 6.15: DMP of PX4 autopilot tasks under dual core processor with different period configurations

$\mathbf{T_1}$	$\mathbf{T_2}$	$\mathbf{T_3}$	DMP τ_1	DMP τ_2	DMP τ_3
4 ms	7 ms	10 ms	0	0.1536	0
3 ms	7 ms	10 ms	2.7×10^{-8}	0.9147	0
3 ms	6 ms	10 ms	2.7×10^{-8}	0.9993	0
3 ms	6 ms	7 ms	2.7×10^{-8}	0.9993	0.0006
3 ms	6 ms	7 ms	2.7×10^{-8}	0.9993	0.0006
2 ms	4 ms	5 ms	0.7082	0.9999	0.9271
4 ms	2 ms	5 ms	0	0	0
3 ms	2 ms	5 ms	5.5×10^{-6}	0	0.0451

We assume that the set of three tasks of the PX4 autopilot (Figure 6.10) is scheduled on a dual core processor with two identical cores. Then, we compute their

DMP to study the schedulability of the system on such hardware architecture. To do so, we apply our partitioning heuristic on these tasks and we assign the sub-tasks **Sensors**, **EKF**, **Nav**, **GYRO** and **GPS** to the first core and the sub-tasks **Pos**, **Att**, **Att_rate** and **Motor_Drv** to the second core.
Results are illustrated in Table 6.15 for different periods combinations. Since priorities of tasks are defined by DM [13], all sub-tasks of task τ_1 have higher priorities than τ_2 in the first six experiment in Table 6.15. We note that DMPs of the three tasks increase as we decrease periods. For the two last experiments, we inverse the priorities of τ_1 and τ_2 by choosing $T2 < T1$. We notice that DMP are significantly reduced even with smaller periods. Thus, we suggest to change the priorities of programs to assign the highest priority to task τ_2. We also note under this configuration, we guarantee low DMPs with smaller periods than the ones obtained in the case of single core. Hence, the parallelization on dual core processor allows to reach a more schedulable and reactive system with higher rates (i.e.smaller periods).

7 Conclusion and Perspectives

Contents

In this chapter, we conclude this thesis. First, we give a summary of our contributions presented in the previous chapters. Second, we propose some research perspectives for future work.

7.1 Contributions

The contributions presented in this manuscript are the following:

- **Deterministic schedulability analysis for the DAG task model:** We proposed sufficient schedulability tests for the DAG task model with deterministic execution and communication times.

    1. We provided methods to compute the response times of DAG tasks based on the characterization of the worst-case arrival patterns of higher priority tasks used in the holistic approach [1].
    2. We also provided two methods to compute the response times based on a new characterization of the worst-case arrival patterns of higher priority tasks.

3. We proved the safety of the response time estimation computed by our proposed methods in polynomial time.

- **Probabilistic schedulability analysis for the DAG task model:**

4. We defined different probabilistic operators (e.g. convolution, maximum) and we employed them to extend our deterministic RTA to deal with probabilistic timing parameters.
5. We used a Bayesian network to model dependencies between different random variables employed in the response time equations. Then, we computed the response time distributions using two different inference methods:
   (a) Exact inference with Variable Elimination.
   (b) Forward sampling.
6. We studied the schedulability of probabilistic task model using a C-space representation of probabilistic timing parameters and schedulability conditions. We combined our deterministic schedulability test with a machine learning based classifier (SVM) in order to estimate the schedulability probability of each DAG task and of the whole system.

- **Scheduling techniques for the DAG task model:**

7. We defined priority at the task level using DM policy [13] and Audsley's algorithm[76] combined with our schedulability test. In so doing, we proved that our schedulability test is compliant (in the sense defined in [101]) with Audsley's algorithm.
8. We also defined priority at the sub-task level using different methods:
   (a) Optimal priority assignment for sub-tasks using an MILP formulation [59].
   (b) Heuristic based priority assignment for sub-tasks.
   (c) Genetic algorithm based solution for sub-task priority assignment.
9. We provided a heuristic for assigning sub-tasks to cores that minimizes communication delays and balances the load between cores.
10. We reduced the size of a DAG while preserving the same precedence constraints structure using two methods:
    (a) Exact solution based on an ILP formulation.

(b) A heuristic based on a greedy approach.

11. We proposed a workflow (Figure 5.13) that integrates the different proposed scheduling techniques together in order to reduce the response time of DAGs.

7.2 Research Perspectives

In this thesis, we provided scheduling and schedulability techniques that rise new challenges and can be extended to other task models. In this section, we present research perspectives for our future work. These perspectives are divided into two categories: (i) Short-term perspectives that consist in an incremental extension or improvement of our proposed scheduling techniques. (ii) Long-term perspectives that consist in more challenging extensions and goals.

Short-Term Perspectives:

Better estimation of the jitter

Based on the task model with a Best Case Execution Time (BCET) for each sub-task, we can compute a lower bound of the Best Case Response Time (BCRT) using similar approaches to the one employed in [122–124]. We suggest adapting these approaches to the DAG task model. Indeed, the BCRT helps us to better estimate the jitter that equal to the difference between the WCRT and BCRT of the predecessors. Moreover, we note that in the case of sporadic arrivals, we can ignore the effect of higher priority tasks when computing the BCRT. Thus, we could use the optimal scheduling of DAGs proposed recently by Baruah [59] in order to compute the exact BCRT of a single partitioned DAG task.

Model dependencies between different DAGs using a Bayesian network

For the sake of simplicity in Section 4.3, we deployed the Bayesian network to model dependencies only between sub-tasks inside the same DAG. After that, we computed the distribution of the response time in isolation (i.e. with no higher priority DAGs). As an extension, we suggest using the Bayesian network to model dependencies between different random variables and sub-tasks belonging to different DAG tasks. Then, we could compute the global response time distribution.

Apply RTA to refine scheduling techniques

In Figure5.13, we described a workflow that explains how to apply the different scheduling techniques (proposed in Chapter 5) before moving to the schedulability analysis step (RTA). In future work, we propose to apply this workflow several times in a loop. After each iteration, we use the response time obtained by the RTA to refine the priority assignment and the partitioning produced by this iteration. Hence, the RTA is used as a cost function (or fitness) to minimize by exploring different possible priority assignments and partitioning. The solution space could be extremely large. However, some techniques of local search like GA could provide promising results.

Actually, we have already used a similar strategy for some of the problems studied in this work. We used the results of the schedulability test to assign priority at the task level based on OPA algorithm [76]. We also used the response time as a fitness function for GA when defining priority at the sub-task level. In future work, we could generalize such an approach for all scheduling steps described by the workflow in Figure5.13.

Long-Term Perspectives:

Prove the safe estimation of the $\mathcal{M}\text{ax}_{\text{Indep}}$ operator

In Section 4.2.2, numerical examples showed that $\mathcal{M}\text{ax}_{\text{Indep}}$ operator never under-approximates the actual distribution of response time (obtained by studying all combinations of execution times). We believe that if we have the comonotonic property between two random variables (that are independent or not), then we could prove that the $\mathcal{M}\text{ax}_{\text{Indep}}$ operator will provide a maximum distribution that never under-approximates the actual one.

Sampling techniques that ensure safe estimation

In Section 4.3, we used the forward sampling technique for Bayesian inference in order to estimate the response time distribution. Moreover, in Section 4.4 , we also used sampling to estimate the DMP in C-space. These sampling techniques do not guarantee a safe estimation. Indeed, they could result in an under-approximation of the actual DMP or response time distribution. In future work, we plan to explore other sampling techniques or propose new sampling techniques that guarantee to never under-approximate the estimated value.

Improve the priority assignment heuristic for sub-tasks

In future work, we aim to explore new ideas to improve the heuristic for priority assignment at the sub-task level proposed in Section 5.1.2.3. For example, we could consider the number of cores used by successor sub-tasks and not only the sum of their execution times. In addition, prioritizing sub-tasks that belong to the critical path could be a good alternative. In the case of partitioned scheduling, we note that the critical path also depends on the sub-tasks allocation and not only on the structure of the graph and the cost of the nodes and of the edges.

Use of proposed probabilistic techniques for global scheduling

In Chapter 4, we proposed some techniques (probabilistic operators, Bayesian network and probabilistic C-space) to analyze the schedulability of a task model characterized by probabilistic timing parameters. In future work, we plan to explore applying these techniques to global scheduling. For example, we believe that a Bayesian network could be used to compute the local release time and the local deadline distributions of each sub-task in a DAG task, similarly to the approach used on deterministic DAGs in [51].

Appendices

A

DAG scheduling with MILP Formulation

A.1 MILP Formulation [59]

Let G be a DAG graph composed of n nodes v_i, $\forall\, i \in \{1, \ldots, n\}$ and partitioned on m cores. Each node has an execution time C_i and a deadline D. The MILP formulation of DAG scheduling problem proposed in [59] introduces several real and integer variables as follows:

- s_i and f_i non-negative real-valued variables that represent respectively the start time and finish time of node v_i.

- x_{ij} and y_{ij} zero-one integer variables defined for each pair of nodes v_i and v_j. x_{ij} is equal to 1 if $s_i \leq s_j$ and 0 otherwise. y_{ij} is equal to 1 if $f_i \leq f_j$ and 0 otherwise.

- c_{ijk} non-negative real-valued variables defined for each 3-tuple of nodes v_i, v_j and v_k. c_{ijk} is equal to C_k if the execution of v_k starts after v_i and finishes before v_j (i.e. $s_i \leq s_k$ and $f_k \leq f_j$). Otherwise, c_{ijk} equals to 0.

The MILP formulation and corresponding constraint are given as follows:

$$
\begin{array}{lll}
\text{minimize} & f_i & \text{if } v_i \text{ is thet sink node} \\
\text{subject to} & s_i + C_i \leq fi & \forall \text{ node } v_i \\
& f_i \leq D & \forall \text{ node } v_i \\
& f_i \leq s_j & \forall \text{ edge } (v_i, v_j) \\
& s_i \geq s_j - M \cdot x_{ij} & \text{if } \pi(v_i) = \pi(v_j) \\
& s_j \geq s_i - M \cdot (1 - x_{ij}) & \text{if } \pi(v_i) = \pi(v_j) \\
& f_i \geq f_j - M \cdot y_{ij} & \text{if } \pi(v_i) = \pi(v_j) \\
& f_j \geq f_i - M \cdot (1 - y_{ij}) & \text{if } \pi(v_i) = \pi(v_j) \\
& x_{ij} = 1 - x_{ji} & \text{if } i \neq j \text{ (optional)} \\
& y_{ij} = 1 - y_{ji} & \text{if } i \neq j \text{ (optional)} \\
& c_{ijk} \geq C_k - M \cdot (2 - x_{ik} - y_{kj}) & \text{if } \pi(v_i) = \pi(v_j) = \pi(v_k) \\
& \sum_k c_{ijk} \leq (f_j - s_i) + M \cdot (2 - x_{ij} - y_{ij}) & \text{if } \pi(v_i) = \pi(v_j)
\end{array}
\tag{A.1}
$$

We denote by M a large positive constant.

Solving this MILP problem allows to find the optimal scheduling of the DAG graph G that minimize its finish time. Indeed the solution of this MILP is a valid assignment of different decision and ordering variables (i.e. $s_i,\ f_i,\ x_{ij},\ y_{ij}$ and c_{ijk}) that respect the previous constraints.

A.2 Specific Case

For some DAG structures and node-to-core mappings, the solution of the previous MILP formulation (Equation A.1) provide an unfeasible scheduling of the DAG graph G. Below, we give an example of a such DAG graph.

Example A.1. *In this example, we present a DAG graph composed of four nodes (Figure A.1 and Table A.1) and we illustrate the obtained solution of the corresponding MILP formulation and constraints (Equation A.1).*

Table A.1: Parameters of nodes described by DAG graph in Figure A.1

Node	**C_i**	**Core**	**Deadline**
v_1	2	π_1	12
v_2	2	π_1	12
v_3	3	π_1	12
v_4	5	π_1	12

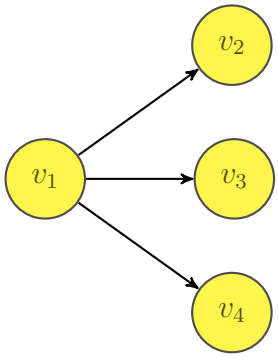


Figure A.1: Example of DAG graph

After using Gurobi solver [105] to solve the MILP formulation (Equation A.1) corresponding to the previous DAG graph (Figure A.1 and Table A.1), we obtain the following assignment for different decision and ordering variable:

$$\{s_i\}_{1\leq i\leq 4} = \{0, 2, 2, 2\}$$

$$\{f_i\}_{1\leq i\leq 4} = \{2, 7, 7, 7\}$$

$$\{x_{ij}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4}} = \begin{pmatrix} 1 & 1 & 1 & 1\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \{y_{ij}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4}} = \begin{pmatrix} 1 & 1 & 1 & 1\\ 0 & 0 & 1 & 1\\ 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=1}} = \begin{pmatrix} 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=2}} = \begin{pmatrix} 0 & 0 & 2 & 2\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=3}} = \begin{pmatrix} 0 & 0 & 0 & 3\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=4}} = \begin{pmatrix} 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 \end{pmatrix}$$

According to the obtained start and finish times and corresponding scheduling, we note that in the time interval $[2,\ 7]$ *core* π_1 *should execute nodes* v_2, v_3 *and* v_4 *with a total execution times equals to* 10 *time units, which is unfeasible because* $10 > 7 - 2$. *Even if we add the optional constraints of MILP formulation (Equation A.1), we obtain similar results. In fact, we introduce two additional constraints* $x_{ij} = 1 - x_{ji}$ *and* $y_{ij} = 1 - y_{ji}$ *when* $i \neq j$. *Otherwise, we set* $x_{ii} = 1$ *and* $y_{ii} = 1,\ \forall\ i \in \{1, \ldots, n\}$. *Hence, we obtain the following solution:*

$$\{s_i\}_{1\leq i\leq 4} = \{0, 2, 2, 2\}$$

$$\{f_i\}_{1\leq i\leq 4} = \{2, 10, 10, 10\}$$

$$\{x_{ij}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4}} = \begin{pmatrix}1&1&1&1\\0&1&0&0\\0&1&1&0\\0&1&1&1\end{pmatrix} \quad \{y_{ij}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4}} = \begin{pmatrix}1&1&1&1\\0&1&0&1\\0&1&1&0\\0&0&1&1\end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=1}} = \begin{pmatrix}2&2&2&2\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}$$

$$\{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=2}} = \begin{pmatrix}0&2&0&2\\0&2&0&2\\0&2&0&2\\0&2&0&2\end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=3}} = \begin{pmatrix}0&3&3&0\\0&0&0&0\\0&3&3&0\\0&3&3&0\end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=4}} = \begin{pmatrix}0&0&5&5\\0&0&0&0\\0&0&0&0\\0&0&5&5\end{pmatrix}$$

We also note that in the time interval $[2,\ 10]$ *core* π_1 *should execute nodes* v_2, v_3 *and* v_4, *which is unfeasible.*

A.3 Adapting Constraints

The problem with the previous DAG graph (Example A.1) comes from the constraints applied on x_{ij} and y_{ij} variables in MILP formulation (from fourth to ninth constraints in Equation A.1). These constraints do not always guarantee that x_{ij} (respectively y_{ij}) variables are equal to 1 when node v_i starts (respectively finishes) before or at the same time as node v_j i.e. $s_i \leq s_j$ (respectively $f_i \leq f_j$). In fact, if $s_i = s_j$ (respectively $f_i = f_j$) then x_{ij} (respectively y_{ij}) could be equal to 1 or 0 while respecting the constraints of the MILP formulation (Equation A.1).

The values of x_{ij} and y_{ij} have an influence on the constraints applied on c_{ijk} variables that should be equal to the execution time C_k if the node v_k is scheduled entirely in the interval $[s_i,\ f_j]$. Consequently, in some cases, the obtained solution of MILP problem schedules in an interval of length l a workload larger than l, which is unfeasible and leads to wrong scheduling.

In order to fix this problem, we add constraints that force x_{ij} (respectively y_{ij}) to be equal to 1 when $s_i = s_j$ (respectively when $f_i = f_j$). These constraints use absolute value operator and they are given as follows:

$$1 - M \cdot |s_i - s_j| \leq x_{ij} \qquad \text{if } \pi(v_i) = \pi(v_j)$$

$$1 - M \cdot |f_i - f_j| \leq y_{ij} \qquad \text{if } \pi(v_i) = \pi(v_j)$$

After using the new constraints, we obtain the following variables assignments:

$$\{s_i\}_{1\leq i\leq 4} = \{0, 5, 2, 7\}$$

$$\{f_i\}_{1\leq i\leq 4} = \{2, 7, 5, 12\}$$

$$\{x_{ij}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4}} = \begin{pmatrix}1&1&1&1\\0&1&0&1\\0&1&1&1\\0&0&0&1\end{pmatrix} \quad \{y_{ij}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4}} = \begin{pmatrix}1&1&1&1\\0&1&0&1\\0&1&1&1\\0&0&0&1\end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=1}} = \begin{pmatrix}2&2&2&2\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}$$

$$\{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=2}} = \begin{pmatrix}0&2&0&2\\0&2&0&2\\0&2&0&2\\0&0&0&0\end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=3}} = \begin{pmatrix}0&3&3&3\\0&0&0&0\\0&3&3&3\\0&0&0&0\end{pmatrix} \quad \{c_{ijk}\}_{\substack{1\leq i\leq 4\\1\leq j\leq 4\\k=4}} = \begin{pmatrix}0&0&0&5\\0&0&0&5\\0&0&0&5\\0&0&0&5\end{pmatrix}$$

From the obtained values of start and finish times of each node, we note that we obtain a feasible scheduling of the DAG graph defined in Figure A.1 and Table A.1. This scheduling is illustrated as follows in Figure A.2:

Figure A.2: Scheduling of DAG graph defined in Figure A.1 and Table A.1 according to the solution of MILP formulation with new constraints.

B

NP-hardness of Graph Reduction Problem

As mentioned in Example 5.8 (on page 131), reducing the size of DAG task requires, in some case, to examine all possible combinations. Hence, it seems to be a NP-hard problem. In order to prove that graph reduction is a NP-Hard problem, we give some hints to reduce the 3-SAT problem (a NP-hard problem) [125] to a graph reduction problem.

Let g be an instance of 3-SAT problem composed of 2 clauses and 3 variables. For example, $g = \underbrace{(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})}_{clause_1} \wedge \underbrace{(x_1 \vee x_2 \vee x_3)}_{clause_2}$.

We could construct a DAG task as in Figure B.1. This DAG is executed on two cores. Each sub-tasks A_i represents a possible assignment for the decision variables of the 3-SAT problem. The sub-tasks that are called $clause_i$ refer to clauses in expression g. Each clause sub-task is connect to all assignment A_i except the assignment that make it equal to 0. The sub-task S is connected to all clause sub-task. S is executed on the same core as all assignment sub-tasks A_i.

If the graph reduction algorithm succeed to merge the sub-task S with any assignment sub-task while preserving precedence structure (i.e. respecting conditions given in Definition 5.1 on page 126), then this assignment makes the expression g equal to 1. In fact, such assignment sub-task is connected to all clause sub-tasks because it is mergeble with the sub-task S that is connected to all clauses. Thus, this assignment makes all clauses equal to 1 and accordingly the expression g is also equal to 1. However, if sub-task S is not mergeable with any assignment sub-task, then there is no valid assignment for x_i's variables that makes the expression g equal to 1.

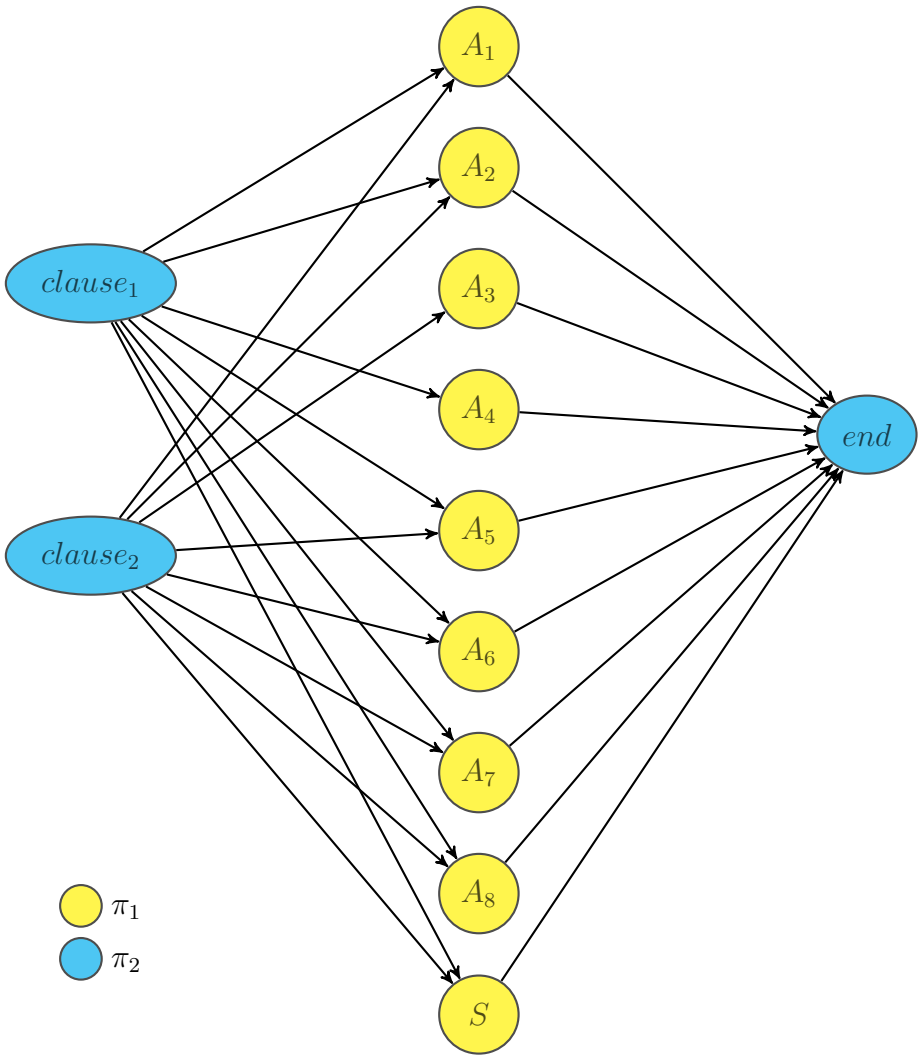


Figure B.1: Example of reducing a 3-SAT problem to a graph reduction problem

C

Example of Generated DAG Tasks

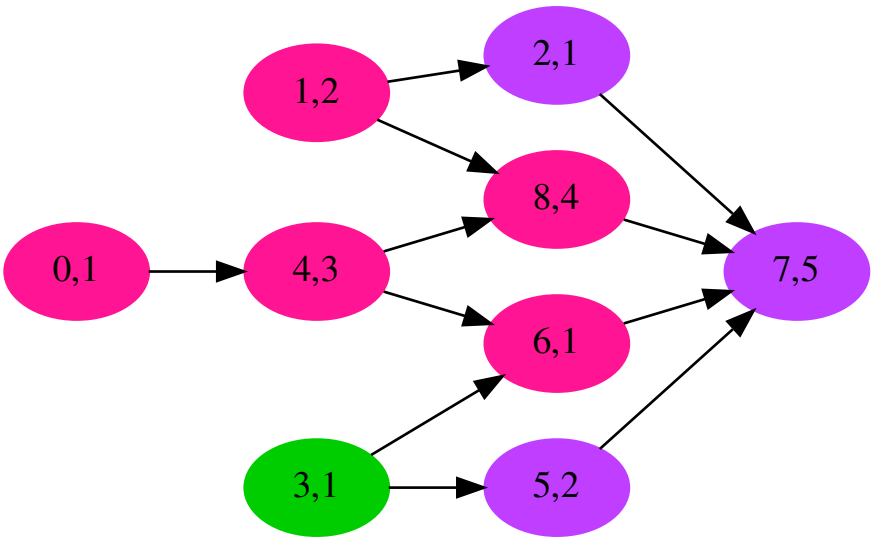


Figure C.1: Example of randomly generated graph partitioned on 3 cores. First number in each node represents its *id* and the second number is its execution time.

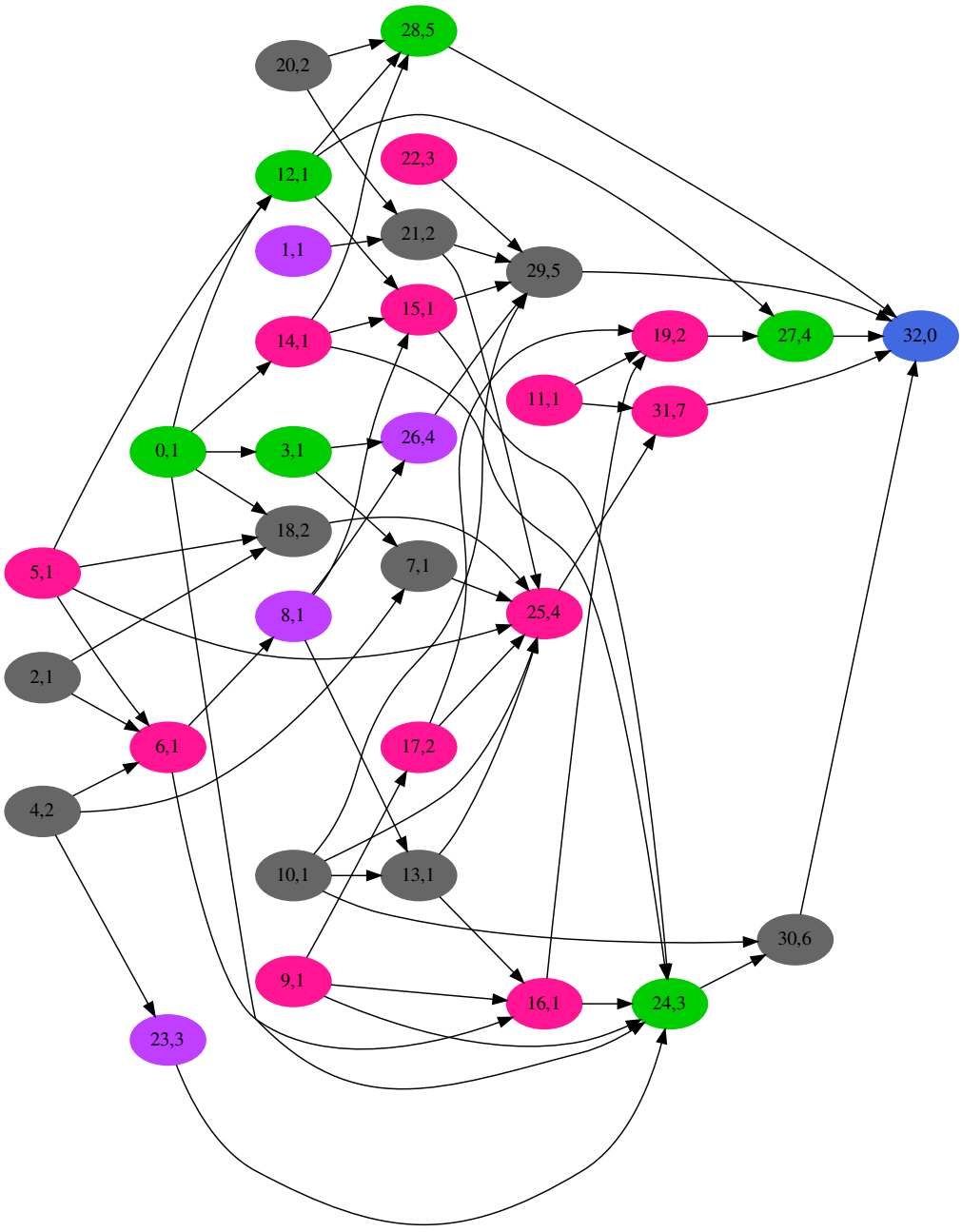


Figure C.2: Example of randomly generated graph partitioned on 4 cores with an additional virtual node (i.e. the blue node with an execution time equal to 0) to ensure a single sink for the graph

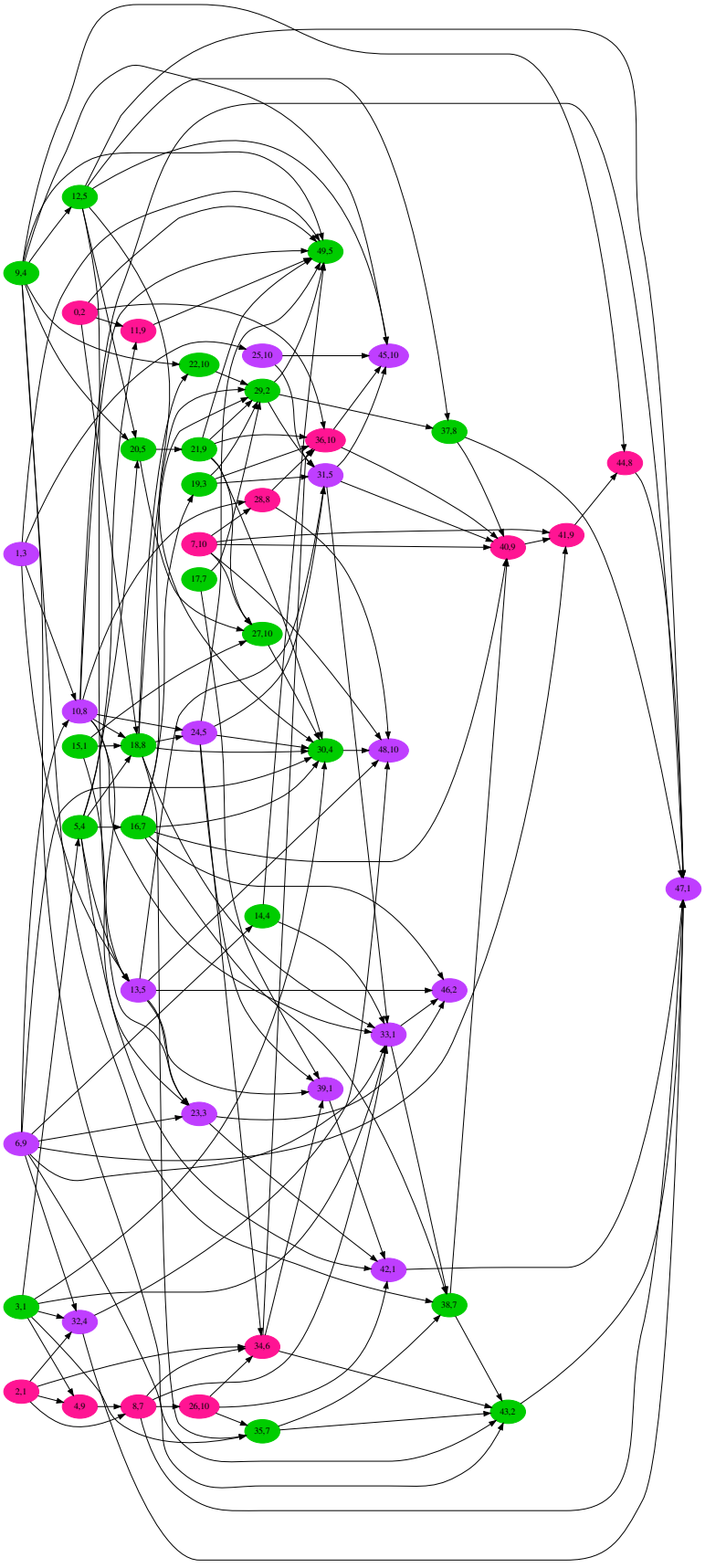


Figure C.3: Example of randomly generated graph partitioned on 3 cores with a probability of creating edges equals to $p = 0.2$.

References

[1] J. C. Palencia Gutierrez, J. J. Gutierrez Garcia, and M. Gonzalez Harbour. "On the schedulability analysis for distributed hard real-time systems". In: *Proceedings Ninth Euromicro Workshop on Real Time Systems*. June 1997, pp. 136–143.

[2] José López et al. "Stochastic analysis of real-time systems under preemptive priority-driven scheduling". In: *Real-Time Systems* 40 (Nov. 2008), pp. 180–207.

[3] Guillem Bernat, Alan Burns, and Martin Newby. "Probabilistic Timing Analysis: An Approach Using Copulas". In: *J. Embedded Comput.* 1.2 (Apr. 2005), pp. 179–194.

[4] J. G. Kassakian et al. "Automotive electrical systems circa 2005". In: *IEEE Spectrum* 33.8 (1996), pp. 22–27.

[5] Intel. *"Intel Xeon Phi Product Family"*. (2016). [Online]. Available: `https://www.intel.com/content/www/us/en/products/processors/xeon-phi.html`.

[6] OpenMP Architecture Review Board. *"API specification for parallel programming OpenMP 5.0"*. (2018). [Online]. Available: `https://www.openmp.org/press-release/openmp-5-0-is-a-major-leap-forward/`.

[7] James Reinders. *Intel Threading Building Blocks*. First. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2007.

[8] Sebastian Altmeyer et al. "A Generic and Compositional Framework for Multicore Response Time Analysis". In: *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. RTNS '15. Lille, France: ACM, 2015, pp. 129–138. URL: `http://doi.acm.org/10.1145/2834848.2834862`.

[9] IEC SC 65A. *Functional safety of electrical/electronic/programmable electronic safety-related systems*. Tech. rep. IEC 61508. 3, rue de Varembé, Case postale 131, CH-1211 Genève 20, Switzerland: The International Electrotechnical Commission, 1998.

[10] ISO-26262. *Road vehicles – Functional safety*. Standard. 2011.

[11] John McDermid and Tim Kelly. "Software in Safety Critical Systems-Achievement & Prediction". In: *Nuclear Future* 2.3 (2006), p. 140.

[12] C. L. Liu and James W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *Journal of the ACM* 20.1 (Jan. 1973), pp. 46–61. URL: `http://doi.acm.org/10.1145/321738.321743`.

[13] Joseph Y.-T. Leung and Jennifer Whitehead. "On the complexity of fixed-priority scheduling of periodic, real-time tasks". In: *Perform. Eval.* 2 (1982), pp. 237–250.

[14] Robert I. Davis and Alan Burns. "A Survey of Hard Real-time Scheduling for Multiprocessor Systems". In: *ACM Comput. Surv.* 43.4 (Oct. 2011), 35:1–35:44. URL: http://doi.acm.org/10.1145/1978802.1978814.

[15] Sanjoy K. Baruah, Louis E. Rosier, and R. R. Howell. "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor". In: *Real-Time Syst.* 2.4 (Oct. 1990), pp. 301–324. URL: https://doi.org/10.1007/BF01995675.

[16] Zonghua Gu et al. "A model-checking approach to schedulability analysis of global multiprocessor scheduling with fixed offsets". In: *Int. J. Embed. Syst.* 6 (2014), pp. 176–187.

[17] Annie Choquet-Geniet and Emmanuel Grolleau. "Minimal schedulability interval for real-time systems of periodic tasks with offsets". In: *Theor. Comput. Sci.* 310 (2004), pp. 117–134.

[18] Joël Goossens, Emmanuel Grolleau, and Liliana Cucu-Grosjean. "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms". In: *Real-Time Systems* 52.6 (Nov. 2016), pp. 808–832.

[19] S. Baruah and A. Burns. "Sustainable Scheduling Analysis". In: *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. Dec. 2006, pp. 159–168.

[20] John Carpenter et al. "A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms". In: *Handbook of Scheduling*. 2004.

[21] S.K. Dhall and C.L. Liu. "On a Real-Time Scheduling Problem". In: *Operations Research* 26.1 (1978), pp. 127–140.

[22] C.A. Phillips et al. "Optimal time-critical scheduling via resource augmentation". In: *the ACM Symposium on theory of Computing*. 1997.

[23] Fisher N. "The multiprocessor real-time scheduling of general task systems". PhD thesis. The University of North Carolina at Chapel Hill, 2007.

[24] J. Anderson and A. Srinivasan. "Early-release fair scheduling". In: *the Euromicro Conference on Real-Time Systems*. 2000.

[25] H. CHO, B. RAVINDRAN, and E.D. JENSEN. "An Optimal Real-Time Scheduling Algorithm for Multiprocessors". In: *the Real-Time Systems Symposium*. 2006.

[26] S. Funk and V. Nadadur. "LRE-TL: An Optimal Multiprocessor Algorithm for Sporadic Task Sets". In: *the Real-Time Networks and Systems conference (RTNS)*. 2009, pp. 159–168.

[27] P. Regnier et al. "RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor". In: *2011 IEEE 32nd Real-Time Systems Symposium*. 2011, pp. 104–115.

[28] B. Andersson, S. Baruah, and J. Jonsson. "Static-Priority Scheduling on Multiprocessors". In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium*. RTSS '01. USA: IEEE Computer Society, 2001, p. 93.

[29] L. Lundberg. "Analyzing Fixed-Priority Global Multiprocessor Scheduling". In: *the Real- Time and Embedded Technology and Applications Symposium*. 2002.

[30] M. Bertogna, M. Cirinei, and G. Lipari. "New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors". In: *the 9th International Conference on Principles of Distributed Systems (ICPDS)*. 2005.

[31] A. Burchard et al. "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems". In: *IEEE Transactions on Computers* 44.12 (1995).

[32] B. Andersson and E. Tovar. "Multiprocessor Scheduling with Few Preemptions". In: *the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2006.

[33] S. Kato and N. Yamasaki. "Portioned EDF-based Scheduling on Multiprocessors". In: *EMSOFT*. 2008, pp. 139–148.

[34] S. Kato, N. Yamasaki, and Y. Ishikawa. "Semi-Partitioned Scheduling of Sporadic Task Systems on Multiprocessors". In: *the Euromicro Conference on Real-Time Systems (ECRTS)*. 2009, pp. 249–258.

[35] K. LAKSHMANAN, R. RAJKUMAR, and J. LEHOCZKY. "Partitioned Fixed-Priority Preemptive Scheduling for Multi-core Processors". In: *the Euromicro Conference on Real-Time Systems (ECRTS)*. 2009, pp. 239–248.

[36] H. LEONTYEV and J.H. ANDERSON. "Hierarchical Multiprocessor Bandwidth Reservation Scheme with Timing Guarantees". In: *the Euromicro Conference on Real-Time Systems*. 2008.

[37] I. SHIN, A. EASWARAN, and I. LEE. "Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors". In: *the Euromicro Conference on Real-Time Systems*. 2008, pp. 181–190.

[38] E.L. Lawler. "Recent results in the theory of machine scheduling". In: (1983).

[39] J. Blazewicz. "Scheduling dependent tasks with different arrival times to meet deadlines". In: (1976).

[40] M. G. Harbour, M. H. Klein, and J. P. Lehoczky. "Fixed Priority Scheduling Periodic Tasks with Varying Execution Priority". In: *the 12th IEEE Real-Time Systems Symposium (RTSS)*. 1991, pp. 116–128.

[41] Houssine Chetto, Maryline Silly, and T. Bouchentouf. "Dynamic Scheduling of Real-Time Tasks under Precedence Constraints". In: *Real-Time Systems* 2.3 (1990), pp. 181–194.

[42] Houssine Chetto and Maryline Chetto. "Scheduling periodic and sporadic tasks in a real-time system". In: *Information Processing Letters* 30.4 (1989), pp. 177–184.

[43] M.Richard et al. "Contraintes de Precedences et Ordonnancement Monoprocesseur". In: *the 10th RTS Embedded Systems*. 2002, pp. 121–138.

[44] L. Cucu and Y. Sorel. "Schedulability condition for real-time systems with precedence and periodicity constraints, without preeemption". In: *the 11th RTS Embedded Systems(RTS)*. 2003.

[45] M. Stigge et al. "The Digraph Real-Time Task Model". In: *the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2011, pp. 71–80.

[46] Karthik Lakshmanan, Shinpei Kato, and Ragunathan (Raj) Rajkumar. "Scheduling Parallel Real-Time Tasks on Multi-core Processors". In: *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*. RTSS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 259–268. URL: https://doi.org/10.1109/RTSS.2010.42.

[47] A. Saifullah et al. "Multi-core Real-Time Scheduling for Generalized Parallel Task Models". In: *IEEE Real-Time Systems Symposium*. 2011.

[48] Ken Tindell and John Clark. "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems". In: *Microprocessing and Microprogramming* 40.2–3 (Apr. 1994), pp. 117–134.

[49] M. Gonzalez Harbour et al. "MAST: Modeling and analysis suite for real time applications". In: *Proceedings 13th Euromicro Conference on Real-Time Systems*. June 2001, pp. 125–134.

[50] Sanjoy Baruah. "Improved Multiprocessor Global Schedulability Analysis of Sporadic DAG Task Systems". In: *Proceedings of the 2014 Agile Conference*. AGILE '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 97–105. URL: https://doi.org/10.1109/ECRTS.2014.22.

[51] Manar Qamhieh et al. "Global EDF Scheduling of Directed Acyclic Graphs on Multiprocessor Systems". In: *Proceedings of the 21st International Conference on Real-Time Networks and Systems*. RTNS '13. Sophia Antipolis, France: ACM, 2013, pp. 287–296. URL: http://doi.acm.org/10.1145/2516821.2516836.

[52] José Fonseca, Geoffrey Nelissen, and Vincent Nélis. "Improved Response Time Analysis of Sporadic DAG Tasks for Global FP Scheduling". In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. RTNS '17. Grenoble, France: ACM, 2017, pp. 28–37. URL: http://doi.acm.org/10.1145/3139258.3139288.

[53] Q. He et al. "Intra-Task Priority Assignment in Real-Time Scheduling of DAG Tasks on Multi-Cores". In: *IEEE Transactions on Parallel and Distributed Systems* 30.10 (2019), pp. 2283–2295.

[54] Jing Li et al. "Outstanding Paper Award: Analysis of Global EDF for Parallel Tasks". In: *Euromicro Conference on Real-Time Systems*. July 2013, pp. 3–13.

[55] José Carlos Fonseca et al. "Response time analysis of sporadic DAG tasks under partitioned scheduling". In: *11th IEEE Symposium on Industrial Embedded Systems, SIES*. 2016, pp. 290–299.

[56] Geoffrey Nelissen et al. "Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks". In: *2015 27th Euromicro Conference on Real-Time Systems* (2015), pp. 80–89.

[57] D. Casini et al. "Partitioned Fixed-Priority Scheduling of Parallel Tasks Without Preemptions". In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. 2018, pp. 421–433.

[58] Hamza Rihani et al. "Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor". In: *RTNS*. Brest, France, Nov. 2016. URL: https://hal.archives-ouvertes.fr/hal-01406145.

[59] Sanjoy Baruah. "Scheduling DAGs When Processor Assignments Are Specified". In: *Proceedings of the 28th International Conference on Real-Time Networks and Systems*. RTNS 2020. Paris, France: Association for Computing Machinery, 2020, pp. 111–116.

[60] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman and Co., 1990.

[61] Stuart Coles. *An introduction to statistical modeling of extreme values*. Springer Series in Statistics. London: Springer-Verlag, 2001.

[62] Paul Embrechts, Thomas Mikosch, and Claudia Klüppelberg. *Modelling Extremal Events: For Insurance and Finance*. Berlin, Heidelberg: Springer-Verlag, 1997.

[63] S. Byhlin et al. "Applying static WCET analysis to automotive communication software". In: *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*. 2005, pp. 249–258.

[64] Stewart Edgar and Alan Burns. "Statistical Analysis of WCET for Scheduling". In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium*. RTSS '01. USA: IEEE Computer Society, 2001, p. 215.

[65] Emil Julius Gumbel. *Statistics of extremes*. Courier Corporation, 2012.

[66] Jeffery Hansen, Scott A Hissam, and Gabriel A Moreno. "Statistical-Based WCET Estimation and Validation." In: *Proceedings of the 9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*. Vol. 10. Jan. 2009.

[67] L. Cucu-Grosjean et al. "Measurement-Based Probabilistic Timing Analysis for Multi-path Programs". In: *2012 24th Euromicro Conference on Real-Time Systems*. 2012, pp. 91–101.

[68] Y. Lu et al. "A Statistical Response-Time Analysis of Real-Time Embedded Systems". In: *2012 IEEE 33rd Real-Time Systems Symposium*. 2012, pp. 351–362.

[69] Yue Lu et al. "A New Way about Using Statistical Analysis of Worst-Case Execution Times". In: *SIGBED Rev.* 8.3 (Sept. 2011), pp. 11–14. URL: https://doi.org/10.1145/2038617.2038619.

[70] J. P. Lehoczky. "Real-Time Queueing Theory". In: *Proceedings of the 17th IEEE Real-Time Systems Symposium*. RTSS '96. USA: IEEE Computer Society, 1996, p. 186.

[71] T.S. Tia et al. "Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times". In: *the 2nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS95)*. 1995, pp. 164–174.

[72] Too-Seng Tia et al. "A linear-time optimal acceptance test for scheduling of hard real-time tasks". In: *Urbana* 51 (1994), p. 61801.

[73] J. L. Diaz et al. "Stochastic analysis of periodic real-time systems". In: *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.* 2002, pp. 289–300.

[74] D. Maxim and L. Cucu-Grosjean. "Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters". In: *2013 IEEE 34th Real-Time Systems Symposium*. 2013, pp. 224–235.

[75] Neil C Audsley. *Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times*. Tech. rep. YCS-164. Department of Computer Science, University of York, 1991.

[76] N.C. Audsley. "On priority assignment in fixed priority scheduling". In: *Information Processing Letters* 79.1 (2001), pp. 39–44. URL: http://www.sciencedirect.com/science/article/pii/S0020019000001654.

[77] S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean. "Schedulability analysis of dependent probabilistic real-time tasks". In: *the 24th International Conference on Real-Time Networks and Systems*. 2016, pp. 99–107.

[78] S. Manolache. "Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour". PhD thesis. Linköping University, Sweden, 2005.

[79] A. M. Leulseged and N. Nissanke. "Probabilistic Analysis of Multi-processor Scheduling of Tasks with Uncertain Parameters". In: *Real-Time and Embedded Computing Systems and Applications, the 9th International Conference, RTCSA*. Vol. 2968. Lecture Notes in Computer Science. 2003, pp. 103–122.

[80] J. Li et al. "Federated scheduling for stochastic parallel real-time tasks". In: *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*. 2014, pp. 1–10.

[81] H. Topcuoglu, S. Hariri, and Min-You Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing". In: *IEEE Transactions on Parallel and Distributed Systems* 13.3 (2002), pp. 260–274.

[82] Yu-Kwong Kwok and Ishfaq Ahmad. "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors". In: *ACM Comput. Surv.* 31.4 (Dec. 1999), pp. 406–471. URL: https://doi.org/10.1145/344588.344618.

[83] E. Bini, M. Di Natale, and G. Buttazzo. "Sensitivity analysis for fixed-priority real-time systems". In: *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*. 2006, pp. 10–22.

[84] Laurent George and Jean-François Hermant. "Characterization of the Space of Feasible Worst-Case Execution Times for Earliest-Deadline-First Scheduling". In: *Journal of Aerospace Computing, Information, and Communication* 6.11 (2009), pp. 604–623. eprint: https://doi.org/10.2514/1.44721. URL: https://doi.org/10.2514/1.44721.

[85] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152. URL: https://doi.org/10.1145/130385.130401.

[86] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Mach. Learn.* 20.3 (Sept. 1995), pp. 273–297. URL: https://doi.org/10.1023/A:1022627411411.

[87] R. I. Davis and L. Cucu-Grosjean. "A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems". In: *LITES* 6.1 (2019), 03:1–03:60.

[88] J. L. Diaz et al. “Pessimism in the stochastic analysis of real-time systems: concept and applications”. In: *25th IEEE International Real-Time Systems Symposium*. 2004, pp. 197–207.

[89] Professor Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. 1st edition. USA: Cambridge University Press, 2009.

[90] Gregory F. Cooper. “The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks (Research Note)”. In: *Artif. Intell.* 42.2–3 (Mar. 1990), pp. 393–405. URL: https://doi.org/10.1016/0004-3702(90)90060-D.

[91] J Pearl. “Fusion, Propagation, and Structuring in Belief Networks”. In: *Artif. Intell.* 29.3 (Sept. 1986), pp. 241–288. URL: https://doi.org/10.1016/0004-3702(86)90072-X.

[92] Max Henrion. “Propagating uncertainty in bayesian networks by probabilistic logic sampling”. In: *UAI ’86: Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence, University of Pennsylvania, Philadelphia, PA, USA, August 8-10, 1986*. Ed. by John F. Lemmer and Laveen N. Kanal. Elsevier, 1986, pp. 149–164.

[93] Haipeng Guo and William Hsu. “A Survey of Algorithms for Real-Time Bayesian Network Inference”. In: (Oct. 2002).

[94] Luca Santinelli. “How effective is sensitivity analysis with probabilistic models?” In: *9th International Real-Time Scheduling Open Problems Seminar, RTSOPS 2018*. 2018, pp. 5–6.

[95] R. L. Wilder. “Evolution of the Topological Concept of "Connected"”. In: *The American Mathematical Monthly* 85.9 (1978), pp. 720–726. URL: http://www.jstor.org/stable/2321676.

[96] James R Munkres. *Topology*. 2nd edition. Prentice Hall Upper Saddle River, NJ, 2000.

[97] Stephen V. Stehman. “Selecting and interpreting measures of thematic classification accuracy”. In: *Remote Sensing of Environment* 62.1 (Oct. 1997), pp. 77–89.

[98] Leonard Kleinrock. “Analysis of A time-shared processor†”. In: 1964.

[99] J. Nagle. “On Packet Switches with Infinite Storage”. In: *IEEE Transactions on Communications* 35.4 (Apr. 1987), pp. 435–438.

[100] Robert Davis and Alan Burns. “Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems”. In: *2009 30th IEEE International Real-Time Systems Symposium RTSS’09*. 2009, pp. 398–409.

[101] Robert Davis and Alan Burns. “Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems”. In: *Real-Time Systems* 47 (Sept. 2011), pp. 1–40.

[102] J. D. Ullman. “NP-Complete Scheduling Problems”. In: *Journal of Computer and System Sciences* 10.3 (June 1975), pp. 384–393.

[103] A. B. Kahn. “Topological Sorting of Large Networks”. In: *Commun. ACM* 5.11 (Nov. 1962), pp. 558–562. URL: http://doi.acm.org/10.1145/368996.369025.

[104] IBM. *"CPLEX Optimizer"*. (2020). [Online]. Available: https://www.ibm.com/fr-fr/analytics/cplex-optimizer.

[105] Gurobi Optimization. *"Gurobi Optimizer 9.0 release"*. (2020). [Online]. Available: https://www.gurobi.com/products/gurobi-optimizer/.

[106] Inman Harvey. "The Microbial Genetic Algorithm". In: *Proceedings of the 10th European Conference on Advances in Artificial Life: Darwin Meets von Neumann - Volume Part II*. ECAL'09. Budapest, Hungary: Springer-Verlag, 2009, pp. 126–133.

[107] J. Ullman. "The performance of a memory allocation algorithm". In: 1971.

[108] David Johnson. "Near-optimal bin packing algorithms". PhD thesis. Massachusetts Institute of Technology, 1973.

[109] M.R Garey et al. "Resource constrained scheduling as generalized bin packing". In: *Journal of Combinatorial Theory, Series A* 21.3 (1976), pp. 257–298.

[110] Thomas L. Adam, K. M. Chandy, and J. R. Dickson. "A Comparison of List Schedules for Parallel Processing Systems". In: *Commun. ACM* 17.12 (Dec. 1974), pp. 685–690.

[111] Edward Grady Coffman and John L Bruno. *Computer and job-shop scheduling theory*. 1st edition. John Wiley & Sons, 1976.

[112] "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey". In: *Discrete Optimization II*. Ed. by P.L. Hammer, E.L. Johnson, and B.H. Korte. Vol. 5. Annals of Discrete Mathematics. Elsevier, 1979, pp. 287–326.

[113] Hamid Reza Boveiri. "Task Assigning Techniques for List-Scheduling in Homogeneous Multiprocessor Environments: A Survey". In: *International Journal of Software Engineering and its Applications* 9 (Dec. 2015), pp. 303–312.

[114] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. "SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms". In: *Proc. of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*. WATERS. 2014.

[115] P. Emberson, R. Stafford, and R.I. Davis. "Techniques For The Synthesis Of Multiprocessor Tasksets". In: *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. July 2010, pp. 6–11.

[116] Enrico Bini and Giorgio C. Buttazzo. "Measuring the Performance of Schedulability Tests". In: *Real-Time Systems* 30.1–2 (May 2005), pp. 129–154.

[117] R. I. Davis, A. Zabos, and A. Burns. "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems". In: *IEEE Transactions on Computers* 57.9 (2008), pp. 1261–1276.

[118] Daniel Cordeiro et al. "Random Graph Generation for Scheduling Simulations". In: *the 3rd International ICST Conference on Simulation Tools and Techniques*. 2010, 60:1–60:10.

[119] Takao Tobita and Hironori Kasahara. "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms". In: *Journal of Scheduling* 5 (Sept. 2002), pp. 379–394.

[120] Yu-Kwong Kwok and Ishfaq Ahmad. “Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors”. In: *ACM Comput. Surv.* 31.4 (Dec. 1999), pp. 406–471.

[121] Kasahara and Narita. “Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing”. In: *IEEE Transactions on Computers* C-33.11 (1984), pp. 1023–1029.

[122] J. C. Palencia Gutierrez, J. J. Gutierrez Garcia, and M. Gonzalez Harbour. “Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems”. In: *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No.98EX168)*. 1998, pp. 35–44.

[123] Taewoong Kim et al. “Best Case Response Time Analysis for Improved Schedulability Analysis of Distributed Real-Time Tasks”. In: *ICDCS Workshop on Distributed Real-Time Systems*. 2000.

[124] Reinder J. Bril, Johan J. Lukkien, and Rudolf H. Mak. “Best-Case Response Times and Jitter Analysis of Real-Time Tasks with Arbitrary Deadlines”. In: *Proceedings of the 21st International Conference on Real-Time Networks and Systems*. RTNS ’13. Sophia Antipolis, France: Association for Computing Machinery, 2013, pp. 193–202.

[125] Richard Karp. “Reducibility Among Combinatorial Problems”. In: vol. 40. Jan. 1972, pp. 85–103.