Doctoral Thesis

Probabilistic Analysis and Scheduling of Real-Time Systems

Anna Friebe
School of Innovation, Design and Engineering (IDT)
Mälardalen University

Main Supervisor: **Thomas Nolte**
Co-supervisors: **Alessandro V. Papadopoulos**
Filip Marković

Till Ida, Rasmus och Jimmy.

Den störste fanatikern är den störste tvivlaren. Han vet det inte.
Han är en pakt mellan två
där den ene ska vara synlig till hundra procent och den andre osynlig.
Vad jag avskyr uttrycket ”till hundra procent”!

Tomas Tranströmer

Abstract

In this thesis, probabilistic methods are explored for the analysis and scheduling of real-time systems, where computation times vary significantly. The aim is to enable sufficient timing-related performance while allowing for economic resource provisioning or other average-case objectives. In one line of research, Hidden Markov Models (HMMs) with continuous emission distributions are used to model execution times of periodic tasks. A framework for the identification and validation of such models is presented. Methods are developed for updating model parameters in systems where the execution time behavior changes, and for bounding the deadline miss probability for such periodic tasks in a reservation-based server. For scheduling parallel workload with varying computational demand, a mechanism is proposed for sharing a job queue among several reservation-based servers. The mechanism guarantees executing jobs a certain amount of computational resources prior to their deadline, by enabling job dismissal in overload situations. Another contribution regards parallel synchronous tasks, and the problem of assigning a suitable number of cores to the task, so that the deadline is met while optimizing towards a goal such as minimizing energy consumption. A suitable core assignment is found using a Multi-Armed Bandit (MAB) formulation of the problem, requiring only limited knowledge of the worst-case properties of the task structure. Using derived response time bounds in the MAB formulation reduces the time to convergence and the energy consumption.

Sammanfattning

I den här avhandlingen utforskas probabilistiska metoder för analys och schemaläggning av realtidssystem där beräkningstider varierar väsentligt. Syftet är att möjliggöra tillräcklig tidsrelaterad prestanda och samtidigt tillåta ekonomisk resursförsörjning eller genomsnittsrelaterade mål. I en forskningsgren används Hidden Markov Models (HMMs) med kontinuerliga emissionsdistributioner för att modellera exekveringstider hos periodiska uppgifter. Ett ramverk för att identifiera och validera sådana modeller presenteras. Metoder utvecklas för att uppdatera modellparametrar i system där exekveringstidernas beteende ändras, och för att begränsa sannolikheten att sådana periodiska uppgifter i en reservationsserver missar deadlines. För att schemalägga parallellt arbete med varierande beräkningsmässiga krav föreslås en mekanism för att dela en jobbkö mellan flera reservationsservrar. Mekansimen garanterar exekverande jobb ett visst mått av beräkningsresurs innan sin deadline, genom att avvisa jobb i överbelastningssituationer. För parallela synkrona uppgifter, och problemet att tilldela ett lämpligt antal beräkningskärnor för en uppgift, så att deadline möts och man optimerar mot ett mål såsom att minimera energikonsumtion. En lämplig kärntilldelning hittas genom att använda en Multi-Armed Bandit (MAB) formulering av problemet, som endast kräver begränsad kunskap om egenskaperna hos uppgiftens struktur i värsta fallet. Genom att använda härledda gränser för responstiderna i MAB formuleringen minskar tiden till konvergens och energikonsumtionen.

Acknowledgment

I am so grateful to my supervisors, Thomas Nolte, Alessandro Papadopoulos, and Filip Marković, for their support and patience. Thank you for the inspiring scientific discussions, for your encouragement at times of frustration, and for your invaluable feedback. I express my special thanks to the younger generation of potential scientists who have brought joy to this project, particularly Ruth, Mia, and Leonardo.

The discussions with Tommaso Cucinotta, Alberto Marchetti-Spaccamela, and Sanjoy Baruah have been inspiring, and I greatly appreciate our collaborations. Thank you, Mikael Sjödin, for reviewing the thesis proposal, and I also wish to thank all anonymous reviewers who provided so much valuable feedback during this journey.

I wish to thank all current and former colleagues at MDU for intriguing discussions, laughs, and fika.

Tack till Vetenskapsrådet, KK-stiftelsen och MDU för finansiering av detta projekt.

Jag vill tacka mina vänner som fortfarande finns där, trots min återkommande frånvaro, speciellt Lotta, Carina, Maria, Maria, Lisa, Erik, Marie och Niklas.

Jag vill också tacka mina bröder Anders och Jonas, min kära mamma Monica som alltid sprider glädje, och Mats-Olof. Tack till Jimmy för att jag fått skratta och gråta med dig i mer än halva livet. Ida och Rasmus, det är fantastiskt att få vara mamma till er, ni är det bästa som hänt mig. Tack för att ni står ut.

Anna Friebe, Västerås, March 2025

List of Publications

Papers included in this thesis[1]

Paper A: Anna Friebe, Alessandro V. Papadopoulos, and Thomas Nolte. *Identification and Validation of Markov Models With Continuous Emission Distributions for Execution Times.* In IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2020.

Paper B: Anna Friebe, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte. *Adaptive Runtime Estimate of Task Execution Times Using Bayesian Modeling.* In IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2021.

Paper C: Anna Friebe, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte. *Efficiently Bounding Deadline Miss Probabilities of Markov Chain Real-Time Tasks.* In Real-Time Systems, 2024.

Paper D: Anna Friebe, Tommaso Cucinotta, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte. *Nip it in the Bud: Job Acceptance Multi-Server.* Accepted at IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) 2025.

[1] The included papers have been reformatted to comply with the thesis layout.

Paper E: Anna Friebe, Alberto Marchetti-Spaccamela, Tommaso Cucinotta, Alessandro V. Papadopoulos, Thomas Nolte, and Sanjoy Baruah. *Resource Management for Stochastic Parallel Synchronous Tasks: Bandits to the Rescue.* Under review.

Related publications, not included in this thesis

Anna Friebe and Ronny Eriksson. *Challenges for Autonomous Sailing Robots*, In Proceedings of the 14th Conference on Computer Applications and Information Technology (COMPIT) 2015.

Teo Enqvist, Anna Friebe, and Florian Haug. *Free Rotating Wingsail Arrangement for Åland Sailing Robots*, In Proceedings of the 9th International Robotic Sailing Conference, 2016.

Anna Friebe, Mikael Olsson, Maël Le Gallic, Jordan Less'ard-Springett, Kjell Dahl, and Matias Waller. *A Marine Research ASV Utilizing Wind and Solar Power*, In OCEANS 2017.

Jordan Less'ard-Springett , Anna Friebe, and Maël Le Gallic. *Voter Based Control System for Collision Avoidance and Sailboat Navigation*, In Proceedings of the 10th International Robotic Sailing Conference, 2017.

Anna Friebe, Giammarco René Casanova, Maël Le Gallic, Clement Rolinat, and Matias Waller. *Situational Awareness and Obstacle Avoidance for a Wind Propelled Marine Research ASV* In Proceedings of the 17th Conference on Computer Applications and Information Technology (COMPIT) 2018.

Jonathan Thörn, Najda Vidimlic, Anna Friebe, Alessandro V. Papadopoulos, and Thomas Nolte. *Work-In-Progress: Probabilistic Timing Analysis of a Periodic Task on a Microcontroller* In IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2019.

Anna Friebe, Alessandro V. Papadopoulos, and Thomas Nolte. *Work-In-Progress: Validation of Probabilistic Timing Models of a Periodic Task With Interference – A Case Study* In IEEE Real-Time Systems Symposium (RTSS), 2019.

Manuel F. Silva, Anna Friebe, Benedita Malheiro, Pedro Guedes, Paulo Ferreira, and Matias Waller. *Rigid Wing Sailboats: A State of the Art Survey* In Ocean Engineering, 2019.

Andreas Mäkilä, Anna Friebe, Leif Enblom, Per-Erik Strandberg, and Tiberiu Seceleanu. *A Generic Software Architecture for PoE Power Sourcing Equipment* In Annual Computers, Software, and Applications Conference (COMPSAC), 2022.

Contents

I

Thesis

Chapter 1

Introduction

In real-time systems, timing-related requirements need to be fulfilled in addition to functional requirements. The timing requirements may be related to safety - car brakes need to be applied promptly when the pedal is pushed, and radiation needs to be applied to tumor tissue for a precise amount of time to kill the tumor but spare surrounding areas. Timing requirements may also relate to non-safety-critical operations. For example, a video streaming application that fails to meet timing requirements causes deterioration to the Quality of Service (QoS) [1]. In process or robotic control, timing-related requirements are related to Quality of Control (QoC) [2].

The functionality of a real-time system is implemented as components that we refer to as *tasks*. A task may release several *jobs*, and a job may consist of several threads that allow the work to be performed in parallel. In a *hard real-time system*, every job's deadline must be met. In *soft* and *firm* real-time systems, some jobs may miss their deadlines. In a soft real-time system, the result of a late job is still useful, but in a firm system, the result is of no use after the deadline [3]. Conventional analysis of hard real-time systems estimates or upper bounds the Worst-Case Execution Time (WCET) [4], and Worst-Case Response Time (WCRT) to ensure that computational resources are sufficient and deadlines are always met.

Today's complex computing systems consist of interconnected components. Computing hardware features include pipelines, branch prediction,

out-of-order execution, and caches or scratchpads. Parallel computation may use multiple cores or dedicated hardware such as Graphics Processing Units (GPUs). Most hardware acceleration features significantly improve the average timing behavior but affect the worst-case behavior to a much smaller extent, contributing to variability in the execution times. For these complex computing systems, it becomes difficult to find tight bounds on WCET and WCRT using conventional analysis techniques [5, 6, 7, 8]. Systems may have components with computational demands that change due to interaction with the environment or other components. Components may have different criticality levels, that is, different degrees of assurance against failures [7]. Components may run locally or remotely. Decisions to run components remotely or on special hardware may be taken dynamically during system operation.

In most real-time systems, some jobs can miss their deadline, even jobs of the most critical task [9]. For these systems, the conventional requirement that *every job must meet its deadline* is too restrictive, and an alternative quantification of the timing-related requirement is more useful. This can be in the form of maximum data age or reaction time [10], weakly-hard [11] or probabilistic guarantees [12, 13, 14], or other timing-related performance guarantees [15, 16].

Instead of only considering the worst-case execution time that occurs very rarely, a probabilistic analysis [17, 18] considers the likelihood of different execution times. A useful requirement may be that the probability of missing a deadline is sufficiently low. To ensure this, most jobs must receive computational resources at the time they need them. Suppose a long computation time value is often followed by more computation times longer than average. In that case, more computational resources are required, compared to the case where long computation times are equally likely over time. When a computation time value affects the probability of computation time values of successive jobs, we say that computation times are dependent or correlated. With a stochastic execution time model, assuming independence simplifies the analysis greatly. Unfortunately, this assumption often does not hold; in reality, longer execution times often tend to cluster in time.

Computers with multiple processors increase the computational resources available at a single point in time. If work can be performed in parallel,

the chances of meeting timing requirements increase. When job computation times vary significantly, static job-to-processor assignments may lead to deadline misses due to queued jobs on one processor, although another processor is idle. Scheduling of computation time varying jobs carries a risk of not meeting timing requirements due to overload situations. The risk can be decreased by assigning additional computational resources, with most processors idle for a large part of the time.

The main challenges this thesis addresses are in two lines of research, both focused on tasks with varying job computation times. The first regards capturing the execution time behavior of a task, including dependencies, in a model that generates data with similar characteristics to the task's execution times. This model is used to bound the deadline miss probability of the task. The second regards scheduling parallel workloads on multiprocessor systems. For a soft real-time task that may experience periods when the resource demand exceeds the supply, we address the problem of ensuring scheduled jobs have a given probability of meeting their deadlines at the price of dismissing some jobs. For a task with hard timing constraints and overprovisioned resources, the challenge of optimizing the resource assignment towards a goal such as energy minimization is addressed.

1.1 Thesis Overview

The background for the thesis, for example, task models, scheduling, and probability theory, is provided in Chapter 2. Chapter 3 presents related work with a focus on execution time dependencies, parallel workload scheduling, and energy-aware scheduling. The overarching research goal and the research questions are outlined in Chapter 4. Research methodology is discussed in Chapter 5, along with an introduction to threats to validity. Contributions are described in Chapter 6, and threats to validity are discussed in relation to the contributions. Conclusions and future directions are presented in Chapter 7.

Chapter 2

Background

This section provides background on task models and scheduling, probability theory, Hidden Markov Models (HMMs), and Multi-Armed Bandits (MABs).

2.1 Task Models

The tasks implementing system components have computational requirements at different points in time. We describe this with task models, where a task's computations are performed in one or more jobs. A job arrives at a scheduler at an *arrival time*. In this thesis, no release jitter is assumed - a job is ready to be processed at arrival. The scheduler determines when the job is processed. The time when the job completes processing is the *finishing time*. The time from arrival to finishing time is the job's *response time*. The task models used in this thesis are briefly outlined in this section.

2.1.1 Periodic Task Model

In a periodic task model, a task τ consists of a sequence of jobs. Job arrivals are separated by a period T. Job J_i has the arrival time a_i, and finishing time f_i. Assuming no jitter in arrival times, we have $a_{i+1} = a_i + T$.

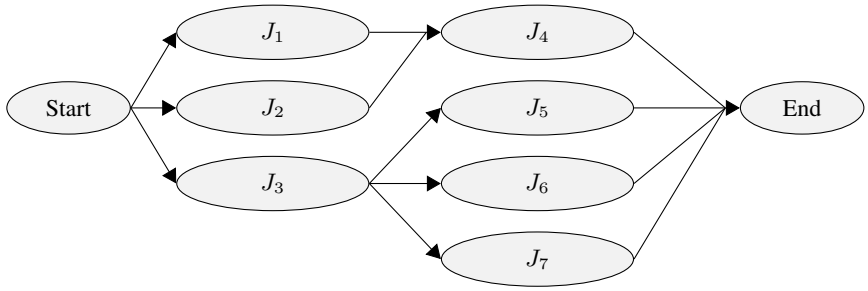


Fig. 2.1: An illustration of a DAG task, where the jobs have precedence constraints. J_j cannot start executing until all jobs with edges directed to J_j have completed.

2.1.2 Task Model with Simultaneous Job Arrival

To better accommodate parallel workload, a more general task model is presented, where at each arrival instant, at most κ jobs from the task τ arrive. Arrival instants are separated by at least p.

2.1.3 Task Models with Precedence Constraints

Precedence constraints imply restrictions on the execution order of computational components - processing one computational component can only start when some other computations are completed. These constraints are often modeled as Directed Acyclic Graphs (DAGs). Such constraints may exist between different tasks [19] or within a single task [20, 21]. Fig. 2.1 illustrates a task with precedence constraints between jobs, as described in [20]. The computations of a job are performed sequentially. The *span* of a DAG task is the sum of the job computation times on the longest path from start to end, and the *work* of the task is the sum of all the job computation times.

The parallel synchronous task model [21] describes a task with jobs that consist of sequential segments, as illustrated in Fig. 2.2. Each segment contains threads that can run in parallel. A segment ends at a synchronization point, so threads in one segment need to be completed before the computation of the next segment can begin.

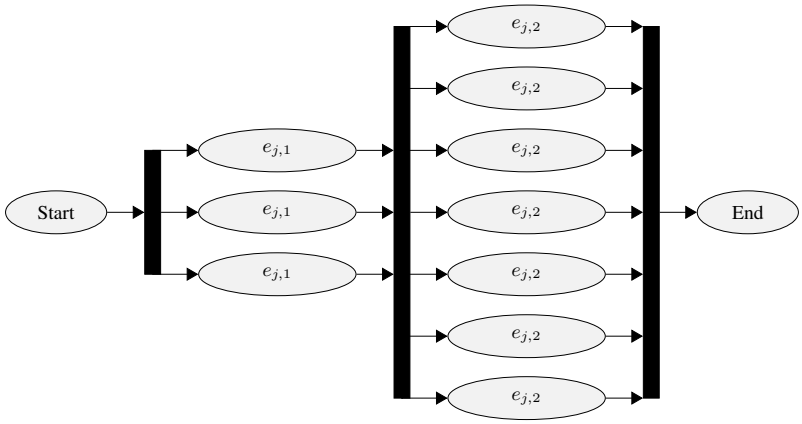


Fig. 2.2: An illustration of a parallel synchronous task τ_j, where the threads in each segment k have a worst case computation time $e_{j,k}$.

Richer DAG-based models include the conditional parallel DAG task model [22] and the multi-DAG model [23]. In the conditional parallel DAG task model, parallel nodes are combined with nodes representing if-then-else clauses. In the multi-DAG model, different execution flows are modeled as separate DAGs.

2.2 Scheduling

Scheduling is the assignment of resources to tasks. Although scheduling may apply to different kinds of resources, in the context of this thesis, we refer to the assignment of computational resources or processor time. A scheduling algorithm determines which jobs or threads to run on a certain processor at a specific time. The jobs that have arrived are ready and can be scheduled.

Fixed-priority scheduling is a scheduling algorithm in which each task has a priority. If several jobs are ready, the scheduler selects them in priority order. In Earliest Deadline First (EDF) scheduling, the job with the closest deadline is selected for scheduling - the priority of a job increases as the deadline approaches. A preemptive scheduling algorithm suspends a running job if a

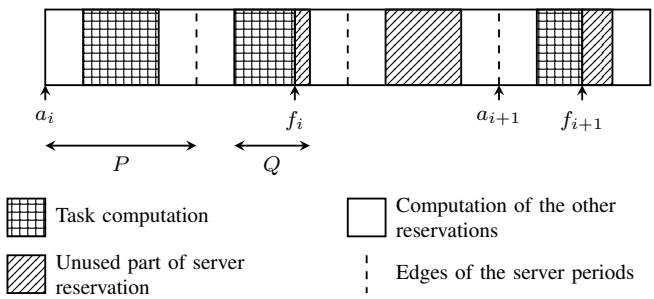


Fig. 2.3: Illustration of a reservation-based server.

higher priority job arrives, to schedule the new job instead.

An important scheduling policy or algorithm property is *sustainability* [24]. If every change in the scheduled task set that is "better" - a computation time is shorter or an interarrival time between jobs is longer - leads to response times that are the same or shorter than prior to the change, then the scheduling policy is sustainable.

The scheduling algorithms used in the thesis are described in the following.

2.2.1 Reservation-Based Scheduling and the Constant Bandwidth Server (CBS)

In reservation-based scheduling, a certain amount of computational resources is reserved for a server within a specific time frame. The server provides the resource to a task or group of tasks. For example, let a server have Q amount of computational resource in each server period P. If a task τ is the only task served by this server, the task will be guaranteed Q amount of computational resource in every time interval of length P, as illustrated in Fig. 2.3.

The advantage of reservation-based scheduling is the *timing isolation* property. A task served by a reservation-based server is guaranteed its budgeted computational resource. If a job in another reservation-based server misbehaves and attempts to use more resources than budgeted, it will not get access to the first task's reservation. In fixed-priority or EDF scheduling, a job that overruns its budget may delay other jobs.

The CBS [25, 26] is a reservation-based server, guaranteeing Q amount of computational resource to a task in every server period of length P, a bandwidth of $\frac{Q}{P}$. A CBS instance tracks the remaining reservation budget q over time and a server deadline d, as illustrated in Fig. 2.4.

When a CBS instance executes on a processor, the remaining budget decreases with the execution time Δt. If the budget reaches 0, the processing is stopped (the server is "throttled") until the server deadline. At this point, the budget is replenished to Q, and the server deadline is increased by P.

Linux provides an implementation of CBS, `SCHED_DEADLINE`[27], allowing for straightforward evaluation of CBS applications.

2.2.2 Graham's List Scheduling

When scheduling jobs or threads with precedence constraints on multiprocessor platforms, Graham's list scheduling [19] is a common and simple algorithm. A priority order (list) is given for the components to schedule. Each time a processor is available, the list is scanned, and the first ready component with fulfilled precedence constraints is selected for processing.

2.3 Probability Theory

To quantify and reason about uncertainty, we apply tools from probability theory. This section introduces necessary concepts without going into the full mathematical formalization. Some inspiration is taken from [28].

Example 2.3.1. *We have two bowls containing red and blue balls. The first bowl has four balls, three red and one blue. The second bowl contains five balls, three blue and two red.*

First, we imagine picking a ball at random from the first bowl, with equal probability of picking each ball. The color of the (imagined) picked ball is a *random variable* that we denote by $\mathcal{X}$. The outcome of $\mathcal{X}$ is one of the values in the sample set $\{b, r\}$ for blue and red. The probability of picking the blue ball is denoted by $\mathbb{P}[\mathcal{X} = b]$ and is $\frac{1}{4}$.

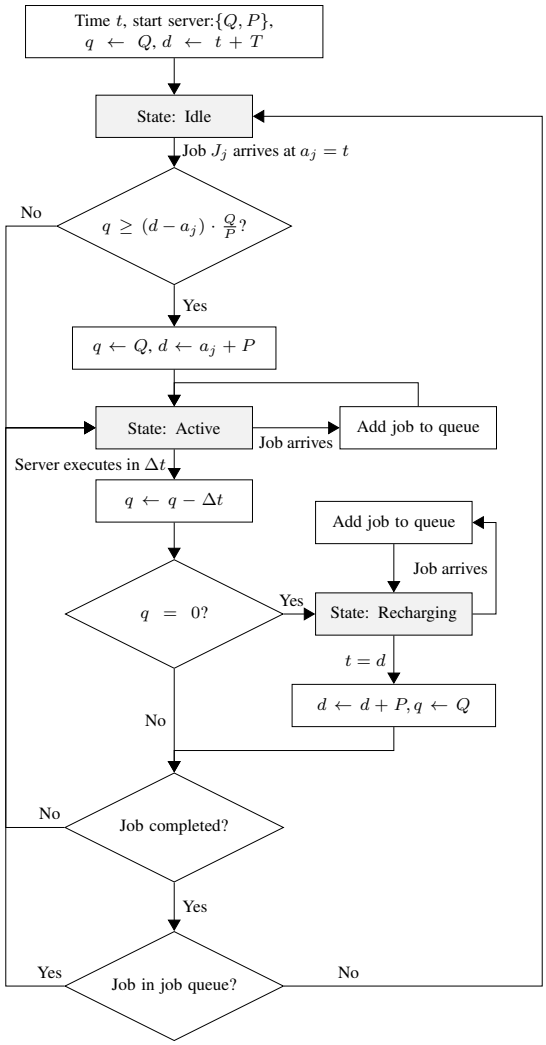


Fig. 2.4: Flowchart of CBS update at runtime.

Second, imagine that we first pick a bowl at random (with equal probability of the bowls), and from that bowl, a ball is randomly picked. The picked bowl is a random variable denoted by $\mathcal{B}$, with the outcome in the sample set $\{1, 2\}$, for the first and second bowl. Let the color of the picked ball in our two-bowl system be the random variable $\mathcal{Y}$. Probability theory allows us to answer questions such as "What is the probability that the selection procedure gives a blue ball?" or "If we picked a blue ball, what is the probability that it came from the second bowl?"

We denote the *joint* probability of a picked ball being blue and coming from bowl 2 as $\mathbb{P}[\mathcal{Y} = b, \mathcal{B} = 2]$. This probability is $\frac{1}{2} \cdot \frac{3}{5} = \frac{3}{10}$, the probability of selecting the second bowl times the probability of a ball being blue in that bowl.

The first question above relates to a *marginal* probability, where we marginalize (sum out) the random variable $\mathcal{B}$. We can obtain this by the *sum rule*: $\mathbb{P}[\mathcal{Y} = b] = \sum_{i=1}^{2} \mathbb{P}[\mathcal{Y} = b, \mathcal{B} = i] = \frac{3}{10} + \frac{1}{8} = \frac{17}{40} = 0.425$.

The second question relates to a *conditional* probability, that we can formalize as $\mathbb{P}[\mathcal{B} = 2|\mathcal{Y} = b]$, the probability that $\mathcal{B}$ equals 2, given that $\mathcal{Y}$ is b. The probability of a ball being blue in the second bowl can be denoted as the conditional probability $\mathbb{P}[\mathcal{Y} = b|\mathcal{B} = 2]$. The *product rule* was used above for the joint probability: $\mathbb{P}[\mathcal{Y} = b, \mathcal{B} = 2] = \mathbb{P}[\mathcal{Y} = b|\mathcal{B} = 2] \cdot \mathbb{P}[\mathcal{B} = 2]$. It also holds that $\mathbb{P}[\mathcal{Y} = b, \mathcal{B} = 2] = \mathbb{P}[\mathcal{B} = 2|\mathcal{Y} = b] \cdot \mathbb{P}[\mathcal{Y} = b]$, which gives us the answer to the second question as $\mathbb{P}[\mathcal{B} = 2|\mathcal{Y} = b] = \frac{\mathbb{P}[\mathcal{Y}=b,\mathcal{B}=2]}{\mathbb{P}[Y=b]} = \frac{3}{10}/\frac{17}{40} = \frac{12}{17} \approx 0.706$.

2.3.1 Discrete and Continuous Random Variables

In Example 2.3.1, the random variables are *discrete*, the balls have a countable number of colors, and there is a countable number of bowls. There are also *continuous* random variables, such as the height of a randomly selected person. In this case, the sample space is an interval of real numbers. In such a case, a choice can be made to divide the interval into a countable number of smaller intervals and use a discrete random variable to represent the interval the person's height falls into.

A Probability Mass Function (PMF) specifies the probabilities of events in the sample space for a discrete random variable. For example, when picking a

ball from the first bowl above, the PMF is defined by:

$$\left(p_X(b) = \mathbb{P}\left[\mathcal{X} = b\right] = \frac{1}{4}, p_X(r) = \mathbb{P}\left[\mathcal{X} = r\right] = \frac{3}{4}\right)$$

The probabilities of all outcomes in the sample space are non-negative and sum to 1.

A probability density function (PDF) $f_X(x)$ specifies for a continuous random variable $\mathcal{X}$ the probabilities of events, such that:

$$\mathbb{P}\left[a \leq \mathcal{X} \leq b\right] = \int_a^b f_X(x)\mathrm{d}x$$

The density $f_X(x) \geq 0, \forall x$, and integrates to 1 over the sample space.

2.3.2 Expected Value

The expected value of a random variable with real-valued outcomes is the weighted mean of the possible outcomes, where the weights are the outcome probabilities. Denote the sample space of $\mathcal{X}$ by Ω. The expected value of $\mathcal{X}$ is denoted $\mathbb{E}[\mathcal{X}]$, and defined as:

$$\mathbb{E}[\mathcal{X}] = \sum_{x \in \Omega} x \cdot \mathbb{P}\left[\mathcal{X} = x\right]$$

For a continuous random variable, the expected value is defined as:

$$\mathbb{E}[\mathcal{X}] = \int_{x \in \Omega} x \cdot f_X(x)$$

2.3.3 Cumulative Distribution Function (CDF) and the Usual Stochastic Order

The CDF $F_X(x)$ of a discrete or continuous random variable $\mathcal{X}$ with real-valued outcome is a function that, for each x, determines the probability that the random variable takes a value lower than or equal to x:

$$F_x(x) = \mathbb{P}\left[\mathcal{X} \leq x\right]$$

In this thesis, when two random variables $\mathcal{X}$ and $\mathcal{Y}$ are compared, the expressions $\mathcal{X} \geq \mathcal{Y}$ or "$\mathcal{X}$ upper bounds $\mathcal{Y}$" refer to the usual stochastic order, according to Definition 2.3.1 restated here from Paper C.

Definition 2.3.1 (cf. [29, 17, 30])**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be two random variables. We say that $\mathcal{X}$ is greater than or equal to $\mathcal{Y}$ (i.e., $\mathcal{X}$ upper bounds $\mathcal{Y}$) if the Cumulative Distribution Function (CDF) of $\mathcal{X}$ is never above that of $\mathcal{Y}$. We denote this relation by $\mathcal{X} \geq \mathcal{Y}$.*

2.3.4 Independent Random Variables

Two events are independent if the joint probability of the events equals the product of the probabilities of each event. In Example 2.3.1, we may consider the two events: "The two-bowl selection procedure gives a blue ball." ($\mathcal{Y} = b$) and "The first bowl is picked." ($\mathcal{B} = 1$). Intuitively, these are not independent events since the color distribution differs in the two bowls. $\mathbb{P}[\mathcal{Y} = b] = 0.425$, as we have seen above, and $\mathbb{P}[\mathcal{B} = 1] = 0.5$, so the product $\mathbb{P}[\mathcal{Y} = b] \cdot \mathbb{P}[\mathcal{B} = 1] = 0.2125$. However, the joint probability of the events is $\mathbb{P}[\mathcal{Y} = b, \mathcal{B} = 1] = 0.5 \cdot 0.25 = 0.125$, as one of the four balls in the first bowl is blue. As they are not equal, the events are dependent.

Two random variables $\mathcal{X}$ and $\mathcal{Y}$ with real-valued outcomes are independent if the joint CDF $\mathbb{P}[\mathcal{X} \leq x, \mathcal{Y} \leq y] = \mathbb{P}[\mathcal{X} \leq x] \cdot \mathbb{P}[\mathcal{Y} \leq y], \forall x, y$.

In the discussion above on Example 2.3.1, the random variable $\mathcal{Y}$ takes the values $\{b, r\}$. Map b to 0 and r to 1, so that $\mathcal{Y}$ instead takes the values $\{0, 1\}$ and is real-valued. $\mathbb{P}[\mathcal{Y} \leq 0] = 0.425$ and $\mathbb{P}[\mathcal{B} \leq 1] = 0.5$, and the product $\mathbb{P}[\mathcal{Y} \leq 0] \cdot \mathbb{P}[\mathcal{B} \leq 1] = 0.2125$. The joint CDF at the same point $\mathbb{P}[\mathcal{Y} \leq 0, \mathcal{B} \leq 1] = 0.125 \neq 0.2125$, so the random variables are dependent.

2.4 Hidden Markov Models

A Markov Model describes a system alternating between different states. This model is *memoryless* - this means that if the system is in state S1 at time $t - 1$, the probability that it is in state S2 at time t is determined by the *transition probability* from state S1 to S2. A three-state Markov Model is illustrated in Fig. 2.5, and a possible state sequence for the model is illustrated in Fig. 2.6.

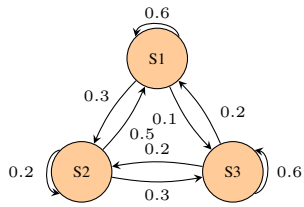


Fig. 2.5: A three-state Markov Model with transition probabilities.

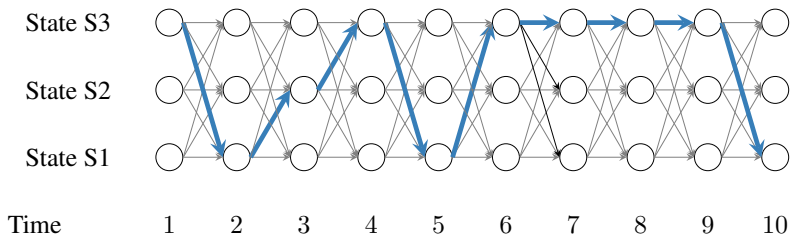


Fig. 2.6: A possible state sequence from the Markov Model in Fig. 2.5.

In a Hidden Markov Model (HMM), the states are not directly observable, but some property related to the states is observed. In papers A-C, task execution times are modeled as an HMM, where different states are associated with different probability distributions over execution times. For example, a periodic task τ is described by the model in Fig. 2.5. Between each job release, the state changes according to the transition probabilities. Different states are characterized by different job execution times, described by probability distributions and referred to as *emission distributions*. In our example task, let the states be associated with Gaussian probability distributions of execution times. The mean and standard deviation for S1 are 20 and 2, for S2 they are 30 and 3, and for S3 they are 40 and 4. One possible execution time realization of the state sequence in Fig. 2.6 is shown in Fig. 2.7.

The tutorial [31] explains the basics of HMMs and some applications to speech recognition.

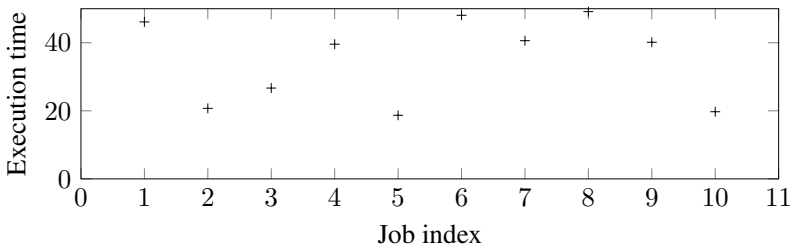


Fig. 2.7: A possible execution time sequence from the state sequence in Fig. 2.6.

2.5 Multi-Armed Bandits

Paper E exploits a Multi-Arm Bandit (MAB) problem formulation to allocate a suitable number of cores to a task. In an MAB problem, an agent repeatedly selects one of k possible actions over T rounds. In each round, the agent receives a reward, and the goal is to maximize the total reward over the T rounds or over the *horizon*. In the standard MAB problem, the k actions or *arms* are associated with fixed probability distributions of rewards unknown to the agent. MAB algorithms make decisions over time under uncertainty and balance *exploration* of arms the agent knows little about with *exploitation* of arms that are likely the best based on observed rewards in previous rounds.

MAB algorithms are often evaluated in terms of *regret*, where the performance of the algorithm is compared to an algorithm that always selects the arm with the highest expected reward. Denote the highest expected reward of an arm by $\rho^{\uparrow}$, and the reward of an action a_i at round i by $\rho(a_i)$. The regret $R(T)$ of the algorithm selecting the actions a_i, restated here from Paper E, is:

$$R(T) = \rho^{\uparrow} \cdot T - \sum_{i=1}^{T} \rho(a_i)$$

Different types of feedback are used in MAB problem formulations. With bandit feedback, the agent only observes the reward from the chosen arm. With complete feedback, the agent can retrospectively observe the reward from all

arms. With partial feedback, the agent receives some additional information beyond the reward of the selected arm.

The agent selects an arm at round i from the information on expected rewards of each arm, often based on the confidence intervals of the expected rewards. For example, an agent using Successive Elimination will remove an arm from the set of possible choices if the upper bound on the confidence interval is below the lower bound on the confidence interval of another arm. An agent using UCB1 will always select the arm with the highest upper confidence bound. This will be either an arm with a high expected reward based on earlier observations, or an arm that has not been explored much, and has a large confidence interval.

The concepts of Bayesian statistics are used in Bayesian bandits. Here, a belief model of the expected arm rewards is maintained. In Thompson sampling, at each round, an arm is selected with the probability that it has the highest expected reward given the belief model. The belief model is updated when feedback is received.

Contextual Multi-Armed Bandit (CMAB) problems include a context. Here, the agent observes some context or feature vector before making the decision, and arms have different reward distributions in different contexts.

In a restless bandit problem [32], the arm reward distributions may change between rounds. In these problems, an infinite horizon is considered, with the objective to maximize the average reward.

Slivkins provides a comprehensive introduction to MABs in [33].

Chapter 3

Related Work

There are several options for quantifying timing-related guarantees in more nuanced ways than the hard real-time guarantee, where *every job must meet its deadline*. Weakly hard real-time systems [11] allow for expressing a minimum required number of met deadlines or consecutively met deadlines in any time window of a determined length, or a maximum number of consecutive deadline misses. The Maximum Reaction Time or Maximum Data Age [10] allows for specifying the maximum time required to complete a cause-effect chain. Control performance and robustness depend on the deadline miss pattern and the control strategy applied at deadline miss, as evaluated in [15, 16]. A Mixed Criticality system [7] has different criticality modes, and tasks in such systems have different criticality levels. The most critical tasks have hard timing constraints. Robustness of Mixed Criticality systems relates to the ability to provide full functionality in the presence of bounded temporal faults, and resilience relates to graceful degradation when temporal faults exceed this bound [7, 34].

In the probabilistic approach, probabilities of events are estimated or bounded. The events may be related to computation times or response times. Specifically, the probability of a job's response time exceeding the relative deadline is the probability of the job missing its deadline [12, 13]. However, more complex events can be considered, such as, for example, the reaction time exceeding some threshold [14]. The surveys by Cucu-Grosjean and Davis on probabilistic timing analysis [17] and schedulability analysis [18] for real-time

systems provide a thorough overview of the field until 2019. Three interpretations of the deadline miss probability concept are outlined in [18]:

1. ”As a probability with a long-run frequency interpretation equating to the expected number of missed deadlines divided by the total number of deadlines in a long (tending to infinite) time interval.

2. As the probability that a randomly selected job will miss its deadline, which is broadly equivalent to the long-run frequency interpretation.

3. As a bound on the probability that any specific job will miss its deadline.”

Interpretations 1 and 2 are broadly equivalent, and in [35] the same concept is referred to as the deadline miss rate, answering the question: ”What is the ratio of jobs missing their deadlines in the long run?”. Interpretation 3 is often referred to as the Worst-Case Deadline Failure Probability (WCDFP) [36, 37, 38].

3.1 Execution Time Dependencies

Erroneously assuming independent execution time random variables in schedulability analysis can cause optimistic results, as noted by Tia *et al.* [12]. Diaz *et al.* [39] first recognized the importance of independence also among job execution times of the same task for their stochastic response time analysis, and noted that the independence assumption is often violated in real-world cases. In [29], the fundamental concept of stochastic pessimism to upper bound distributions according to Definition 2.3.1 was explored.

The pWCET is an upper bound on the job execution time distribution that can be used safely in stochastic response time analysis with independence-assuming tools such as convolution despite potential dependencies [40, 17]. Davis *et al.* [41] highlighted the different interpretations of the uncertainty about the timing behavior of a system. Aleatoric uncertainty stems from the randomness in the system and its environment, while epistemic uncertainty is the lack of complete knowledge about the system in operation. Bozhko *et al.* [42] provided an axiomatic definition of pWCET. This requires conditional independence of jobs' computation times, given a partitioning of the space of

system evolutions, and a probabilistic response time (pRT)-monotonic scheduling policy [42]. A sustainable scheduling policy [24] is pRT-monotonic. The scheduling requirement emphasizes that the pWCET is not a standalone property of a task.

In the literature, several approaches consider probabilistic schedulability analysis with independence assumption for fixed priority, preemptive scheduling, relying on the critical-instant assumption later refuted [43]. Approximations were derived based on queueing theory in [44], and on the Berry-Esseen theorem in [45].

Günzel *et al.* [14] considered probabilistic reaction time analysis for a backlog-free system with partitioned Time-Division Multiple Access (TDMA) scheduling of tasks with independent execution times.

Considering tasks with dependence, copulas were introduced to timing analysis in [46]. Dependencies between random variables were modeled with copulas, transforming the marginal distributions of random variables into a joint distribution. Probabilistic response time bounds were derived for fixed priority preemptive scheduling using known probability distributions, copulas, and Frechet bounds [47].

Extreme Value Theory (EVT) has been applied to address dependence in the areas of measurement-based analysis of execution times [48, 49, 50] and response times [51, 52, 53]. Application of EVT requires that statistical limit laws hold for the sample set [54], and that certain conditions, such as stationarity [55] or extremal independence [56], are fulfilled. As holds for all measurement-based analyses, execution time data used in analysis needs to be representative of execution times in operation [57] to some degree.

The WCDFP for EDF-scheduled tasks considering dependence in bounded intervals was approximated in [36]. Correlation Tolerant Analysis (CTA) using Cantinelli's inequality was presented in [37] to bound WCDFP for fixed priority preemptive scheduled tasks with known bounds on mean and standard deviations of the execution time distributions. In [38], a correlation-aware analysis included bounds on the inter-task and intra-task covariance to derive a tighter bound on the WCDFP compared to CTA.

Mills and Anderson [58] analyzed tasks with stochastic execution times scheduled in servers in a multiprocessor system. Response times and tardiness

were bounded, considering dependence within bounded time windows. Further work [59] provided response time analysis under the assumption that dependence is limited to jobs with execution times below a certain threshold.

Tasks with independent stochastic computation or interarrival times, scheduled in CBS have been analyzed to provide QoS guarantees by deriving the probability of missing the deadline [13], based on the observation that the amount of work waiting to be processed is a Markov chain. In [60], the analysis was extended to stochastic computation and interarrival times. An analytical bound and efficient numerical computation of the probability of deadline miss for periodic tasks were provided in [61]. Dependencies were accounted for by modeling execution times as a Markov chain in [62, 63], deriving the steady state response time distribution and the deadline miss probability with the long-run frequency interpretation.

3.2 Computational Demand Exceeding the Supply

The average available provided resource must exceed the requirement for the system to be stable, that is, response times do not diverge to infinity [25, 39, 44]. Stability is clearly necessary to meet timing requirements. However, it is not sufficient - response times may still be unacceptable if computational demand exceeds the supply in shorter time frames. A system with computational demands exceeding the provided resources may become stable by rejecting jobs entirely or by dismissing jobs, for example, at their deadline, at the cost of some requested work never being performed. A later dismissal point increases the deadline miss rate, as was shown for a uniprocessor system with sufficient average resource provision, and execution times as independent random variables [35]. Schedulability analysis for task graphs with a maximum number of concurrently active graph instantiations was explored in [64]. Rejecting a newly arrived task graph rather than dismissing the oldest instance resulted in a lower number of states and shorter analysis time.

The decision to reject, dismiss, or queue jobs also affects the performance of control systems, as was investigated by Pazzaglia *et al.* [65]. They showed that dismissing jobs at their deadline or rejecting a new job both lead to better control robustness compared to queueing jobs in overload situations.

Mixed Criticality system strategies to ensure that HI-criticality meet their deadlines include abortion or dismissal of LO-criticality jobs [7, 66, 67]. For example, the bailout protocol [68] allows HI-criticality jobs to overrun their normal-mode WCET, and the overrun is compensated for by not starting LO-criticality jobs, and accounting for unused capacity. [69] considered a multiprocessor system with monitoring of HI-criticality jobs. LO-criticality jobs were suspended if a concurrently running HI-criticality job risked overrunning its isolation-based WCET. System engineers have criticized the assumptions of Mixed Criticality systems and argued that LO-criticality tasks should receive some computational resource if at all possible [7, 68]. A robust task can safely drop a non-started job in any extended interval [34]. One of the techniques to achieve resilience is to let any started job run to completion [7].

The Robust Earliest Deadline (RED) algorithm monitors the job queue for a unicore system. Tasks have criticality levels and values, and are scheduled by EDF. A job arrival that leads to a WCET-based overload causes rejection of non-critical jobs based on their values. If processed jobs complete early, rejected jobs may be recovered. Aperiodic tasks in a Total Bandwidth Server (TBS) were scheduled by RED in [70]. To provide QoS guarantees to individual tasks, each task needs to be assigned a separate server [71]. DAG tasks were scheduled holistically in [72]. Instead of dropping the entire DAG when a single node overruns, the node continues to run with slack from nodes completing early or budget from nodes that cannot start due to the overrun.

In the queueing theory literature, different types of dismissal or reneging are analyzed, as surveyed by Ward [73]. It is common to model the probability of clients (jobs) abandoning the queue as a function of the waiting time. Other models include dismissal at a deadline or when a queue is full. Kruk *et al.* [74] analyzed EDF-scheduling in heavy traffic, with jobs dismissed at their deadline. Stochastic knowledge about job deadlines and a finite buffer size was considered in [75]. At buffer overflow, it is at least as good to remove the job stochastically closest to its deadline as an arbitrary job. Different abandonment time and service time distributions at overload were examined in [76]. Mean waiting time and queue length are significantly affected by the abandonment time distribution. Controlling the service rate with a time-varying arrival rate was explored in [77]. Asymptotically stabilizing mean queue lengths and mean

waiting times cannot be achieved simultaneously.

To reduce congestion and waiting time in networking applications, Active Queue Management implies dropping packets before a buffer is full. Random Early Detection [78] uses a probabilistic approach to drop packets, while the more recent CoDel algorithm [79] tracks the minimum waiting time over a time interval. If this exceeds a target value, packets are dropped at an increasing rate until the target waiting time is met.

3.3 Energy-Aware Scheduling

When execution times vary, a real-time system that fulfills needed timing-related guarantees will most likely have a significant amount of unused processor capacity, even if some amount of deadline misses is accepted. The system designer may use such slack to improve the performance (QoS or QoC) by, for example, increasing the frequency or resolution of some computations, or improving the robustness by, for example, adding redundancy. Energy consumption and cost can be reduced by applying dynamic energy management approaches. That is, we optimize towards higher QoS, higher robustness, or lower energy consumption, under the constraints given by the timing-related guarantees. This section focuses on lowering energy consumption, which presents a conflicting objective that is likely bounded by the timing constraints.

The two main technologies to dynamically control energy consumption in modern processors are Dynamic Voltage Frequency Scaling (DVFS) and Dynamic Power Management (DPM). In DVFS, the CPU frequency and voltage are reduced, leading to slower computation at lower power consumption. In DPM, idle processor states are used to reduce power consumption. Deeper idle states turn off more components in the CPU, leading to greater power savings at the cost of longer wake-up latencies.

Energy-aware scheduling surveys are provided in [80, 81, 82]. The focus of [81, 82] is on multicore systems, in [82] mainly DVFS techniques.

Eq. (3.1) describes the power P_{gate} of a single active gate [81], with α as the gate switching probability, C_L the loading capacity, V the supply voltage, f the clock frequency, I_{sc} and I_{leak} the short circuit and leakage currents.

$$P_{gate} = \alpha \cdot C_L \cdot V^2 \cdot f + V \cdot I_{sc} + V \cdot I_{leak} \tag{3.1}$$

The clock frequency reduction enables a reduced supply voltage, which is an important factor in the energy savings from DVFS [83]. The dynamic power component is modeled as $P(f) = \beta \cdot f^{\delta}$, with β as a constant, and $2 \leq \delta \leq 3$ in for example [84, 85]. The voltage scaling window is decreasing in today's low-voltage cores, leading to reduced benefit of DVFS [83]. Additionally, reduced transistor sizes lead to a larger proportion of leakage currents, as these result from a quantum phenomenon [80]. These are two reasons why the energy saving potential of DVFS is decreasing according to [83]. Despite this, DVFS and hybrid techniques remain common in the literature. Recent work in this direction includes [86], where the Adaptive Partitioned EDF scheduler from [87] was extended. Real-time tasks served by CBS were placed on the CPU with the best energy performance on a big.LITTLE architecture, considering timing guarantees, potential frequency changes, and the effect on the whole CPU socket. In [88], the approach was extended for DAG tasks.

The idle states in DPM are often referred to as C-states. C0 is the operational state executing instructions, C1 is a low wake-up latency standby state, and higher numbers refer to sleep states with increasing wake-up latencies and power savings. The C-states of x86 processors were described in [89], along with Package C-states. If all cores on a CPU socket are in a sleep state, a Package C-state is entered, bringing further power savings by partially disabling peripherals. The architecture proposed in [90] retains most of the C-state power savings with significantly lower wake-up latencies.

C-state power management arbiters are considered for energy savings in data centers while meeting service level agreements. In [91], a feedback-based controller determined the minimum number of cores needed to have sufficiently short response times at a given request rate. Idle state residency and network request rate were predicted in [92], and combined with a load balancer, avoiding waking up idle cores if operating cores could accommodate an incoming task. C-states were selected and exited based on the predictions. In [93], the use of C-states was combined with DVFS. When a core was idle, it was always sent to the deepest sleep state. Request tail service time and latency were predicted at request arrivals, to determine a suitable wake-up time and core frequency.

Without explicitly modeling energy consumption, Papadopoulos *et al.* [94] scheduled a DAG task with known worst-case work and span on M identical processors. Initially, the task was assigned $m \leq M$ processors. Based on the worst-case work and span, as well as the number of assigned processors, a virtual deadline was computed. If a job did not finish by the virtual deadline, M processors were assigned at this time. It was shown in [94] that the jobs always meet a hard deadline with this policy. Algorithms were developed to find the lowest m that results in response times below the corresponding virtual deadline. This ensures that the assigned processors are efficiently used, within the hard deadline constraint. The remaining processors can be used for other purposes or put to sleep until the virtual deadline.

Chapter 4

Research Goal and Research Questions

Overarching research goal: To provide tools for analyzing and scheduling real-time systems, where tasks' job execution times vary significantly.

The thesis covers two main lines of research. The first considers a periodic task with dependency between execution times of successive jobs, scheduled on a single processor. The second line of research considers the scheduling of parallel workload on multiple processors.

4.1 Periodic Tasks where Execution Times of Successive Jobs Exhibit Dependence

For periodic tasks with varying computation times and dependency between execution times of successive jobs, the research questions below are investigated. A model capturing the execution time behavior with dependencies allows for a less pessimistic schedulability analysis compared to using the WCET or upper bounding pWCET distribution. **RQ1** relates to identifying an appropriate model and validating it against the observations of the execution time. **RQ2** relates to updating HMM emission distribution parameters if the runtime observations change and become inconsistent with the model.

RQ1: How can we find a model of a task's execution time behavior, including dependencies between successive jobs, so that data generated from the model is consistent with observations?
RQ2: Given an execution time HMM, how can emission distribution parameters be updated at runtime, if observations are no longer consistent with the model?
RQ3: How can we derive a bound on the deadline miss probability of an execution time HMM task in a reservation-based server?

4.2 Scheduling Parallel Workload

Scheduling parallel workload with varying computation times on multiple processors entails additional questions about the number of processors to use and which jobs or threads to execute on which processor. In **RQ4**, a task's jobs may be run in parallel, and the timing requirements are soft - some jobs may miss their deadlines or be discarded. The challenge is to ensure that scheduled jobs are provided with computational resources in time, so that they have a sufficiently high probability of meeting their deadline. In **RQ5**, a parallel synchronous task is considered, with varying thread computation times. A number of processors are reserved for a task, and the challenge is to assign a smaller number of these processors to arriving jobs, which optimizes for an objective such as minimizing the energy consumption. All the reserved processors are assigned at a later point in time if needed to ensure all jobs meet their deadlines.
RQ4: For a task with jobs that may run in parallel, how can started jobs be ensured a determined probability of meeting their deadlines, even if computational demand exceeds the supply?
RQ5: For a parallel synchronous task, how can we learn the number of processors to assign initially, that optimizes for an objective such as energy minimization, while still meeting hard deadlines?

Chapter 5

Research Methodology

The overarching research goal of this thesis is to provide tools for analysis and scheduling of real-time systems, where tasks' job execution times vary significantly. Fig. 5.1 illustrates the research process, reflecting the hypothetico-deductive method [95]. Dashed lines reflect the iterative nature of the process.

Starting from the overarching research goal, the State Of The Art (SOTA) in the literature is reviewed to find interesting research directions to explore further and identify gaps in existing knowledge. Based on the literature review, more concrete problems are formulated to contribute to the overarching goal.

Given the problem formulation, assumptions, and the system model are specified. If applicable, the assumptions and system model are used to derive proofs to address part of the problem. A solution is proposed and implemented, potentially relying on the theoretical results. If there are SOTA solutions suitable for comparison, these are obtained or implemented. Data is collected, applying the different solutions to real and/ or simulated use cases.

In the evaluation, the results from the solutions are compared, and other features of the solutions are discussed. The contributions of the process include published papers with theoretical and empirical results and implemented software used to derive the results.

5.1 Threats to Validity

We consider the validity types discussed in Wohlin et al. [96], namely **conclusion validity**, **internal validity**, **construct validity**, and **external validity**.

Conclusion validity concerns the relation between an intervention and the outcome. Can we be sufficiently sure that differences in results are due to differences in the methods used? Threats to conclusion validity include, for example, small sample sizes, random effects, violation of assumptions of statistical tests, ”fishing” for certain results, and reliability of measures and interventions.

Internal validity concerns issues with experiments where the relation between the intervention and result may be questioned, for example, due to the selection of use cases, tasks, or task sets. The main internal validity issues are related to experiments with subjects such as humans. With human subjects, you cannot apply two different interventions to the exact same subject at the same time and observe the results. Human subjects may be affected by their beliefs about the treatment. These issues do not apply to the research in this thesis, as our subjects are use cases, tasks, and task sets.

Construct validity concerns the generalization of results from an experiment to a concept or theory. One threat to construct validity is insufficiently defined concepts. For example, what does it mean that an intervention is ”better” than another? Other threats to construct validity are related to investigating a single independent variable, a single subject, a single intervention, a single type of measure, or observation. Information loss if a continuous measure is converted to a discrete or binary measure may also be a threat to construct validity.

External validity relates to the generalization of results to, for example, industry practice.

Threats to validity regarding the thesis contributions will be discussed further in Chapter 6.

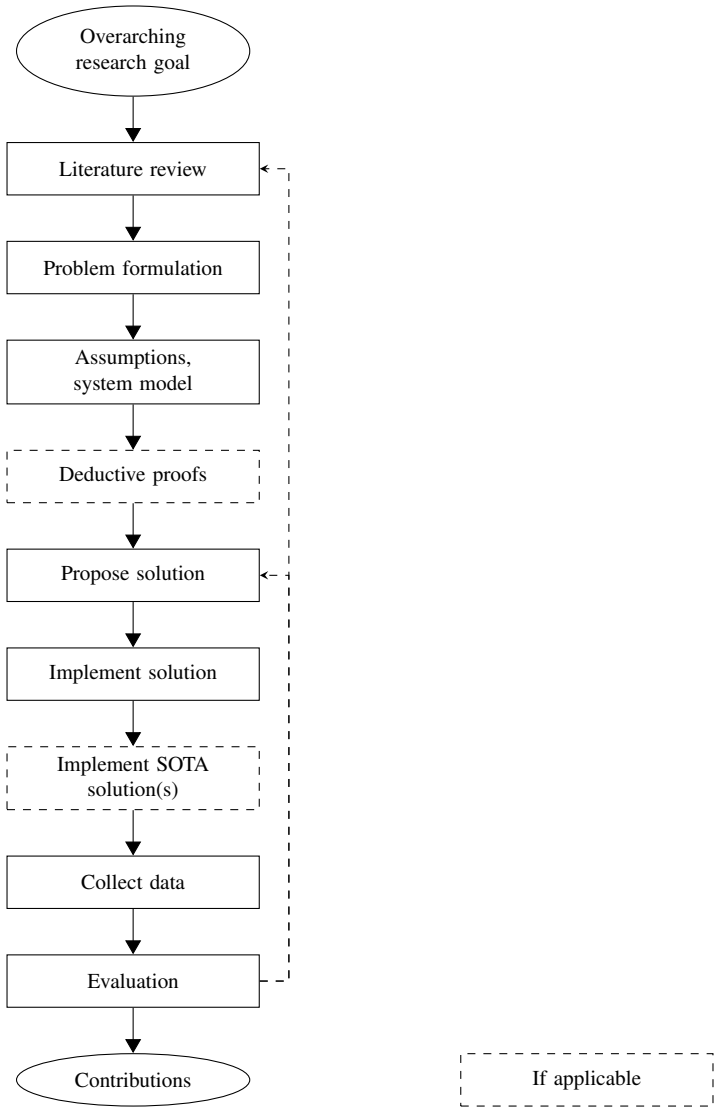


Fig. 5.1: The main steps of the research process.

Chapter 6

Thesis Contributions

In this section, the contributions are outlined, relating to the overarching research goal and the research questions in Chapter 4. Threats to validity are discussed in relation to the different contributions, and the papers included in the thesis are listed.

6.1 Contributions for Periodic Tasks with Execution Time Dependence Between Successive Jobs

In this section, Contributions **C1** through **C3** relating to research questions **RQ1** through **RQ3** are outlined.

C1: A framework for identification and validation of continuous-emission execution time HMMs.

This contribution results from **RQ1**, and is presented in Paper A. An execution time HMM with Gaussian emission distributions is proposed to model the execution time behavior of a task with dependence between successive jobs. A framework is developed for two purposes. The first is to identify such an HMM from execution time traces. The identification process includes automatically determining the number of states, utilizing a tree-based cross-validation approach [97]. The second framework purpose is to validate that execution times generated from the model resemble experimentally obtained execution

times. This is performed using a data consistency approach to model validation [98], comparing data generated from the model with experimental data using a dispersion-based statistic.

The proposed HMM representation is evaluated in two use cases: a simple test program with a known Markov Model structure and a video compression use case. Source code and data are available[1]. The identified HMMs are valid models with respect to the timing trace observations, reinforcing conclusions in previous work [63, 62] that HMMs are useful representations of execution time behavior for some tasks, and extend this to HMMs with Gaussian emission distributions. Preliminary tests show that the identified model in the video decompression test case is invalid for different video inputs. This is consistent with results from [63, 62] that indicate that different inputs correspond to different HMMs and tasks with different resource requirements.

C2: A method for online adaptation of execution time HMM emission distributions.

C2 results from **RQ2**, and is presented in Paper B. In Paper A and in previous work [63, 60], it is shown that different inputs may result in different HMMs. Identification methods for HMMs, as presented in Paper A, are too computationally demanding to perform at runtime. Therefore, in **C2**, adaptation of the emission distributions is explored. In this work, we assume that the number of states and transition probabilities do not change, and that emission distributions change at some instants and remain constant in segments between those instants. In a preprocessing step, the number of states, the transition probabilities, and the emission distribution configurations are identified. The emission distribution configurations are referred to as clusters, and the belief about distribution means and variances is described with a Bayesian approach. The Bayesian cluster description is used to derive a Generalized Likelihood Ratio (GLR) measure, providing a probability that observations in different output segments are generated from the same HMM. At runtime, points of change are detected using the GLR measure. The measure is also used to determine whether a segment is part of an existing cluster, if a new cluster needs to be created, or if two clusters are similar enough to be merged into one cluster. The Bayesian belief about the clusters is updated in the adaptive process. The full

[1] https://github.com/annafriebe/MarkovChainETFramework

adaptive algorithm is compared with two more restrictive versions. In the first, clusters are not created or merged. In the second, clusters are neither created, merged, nor updated after the preprocessing step, so the adaptive process only switches between clusters identified in the preprocessing stage.

Synthetically generated data is used in the evaluation, guaranteeing that assumptions are fulfilled and allowing for comparison to ground truth. Source code and data are available[2]. The Kullback-Leibler (KL) divergence from the predicted Bayesian distribution to the ground truth distribution is calculated to assess the method's performance. When the execution times are generated from an HMM configuration observed during preprocessing, the switching-only adaptive algorithm gives the best result, but for HMM configurations not observed in the preprocessing, the more adaptive algorithms perform better in terms of KL-divergence to the ground truth distribution.

C3: An efficient method for bounding the deadline miss probability of Markov Chain real-time tasks.

C3 addresses **RQ3**, and is presented in Paper C. A task with execution times described by an HMM, scheduled by a reservation-based server, is analyzed. The expected deadline miss probability of a randomly selected job is bounded. The method is based on workload accumulation sequences. Each job arrival results in a specific workload accumulation sequence - the sequence of execution time states since the last idle point. The probabilities of job arrivals resulting in workload accumulation sequences are bounded, starting with short sequences, jobs arriving soon after an idle point. The deadline miss probabilities associated with these workload accumulation sequences are also bounded. The probability of a job arrival resulting in a longer workload accumulation sequence than those accounted for is also bounded. These results are combined, resulting in a bound on the expected deadline miss probability. The bound is updated iteratively by including successively longer workload accumulation sequences. The time required for the bound computation is reduced by combining workload accumulation sequences into order-independent accumulation vectors and by applying a state merging technique.

In the evaluation, a Furuta pendulum [16] control task is analyzed. Empirical deadline miss rates when running the task under the Linux CBS implemen-

[2]https://github.com/annafriebe/AdaptiveETBayes

tation [27] are compared to the discrete-emission distribution HMM deadline miss probability from [63, 99], estimates from simulation, and obtained bounds with and without the state merging technique. Source code and data are available[3]. The proposed method adds some pessimism compared to the discrete-distribution approach. The time required for the bound computation does not depend on the computation time range or the choice of scaling factor, as in the discrete case.

6.2 Contributions for Scheduling Parallel Workload

In this section, contributions **C4** and **C5** addressing research questions **RQ4** and **RQ5** are described.

C4: Job queue sharing by several CBS, along with a job acceptance test.

Paper D presents **C4** resulting from **RQ4**. We consider a task that releases execution time-varying jobs that may run sequentially or in parallel. The number of jobs released at each instant is upper bounded, and there is a minimum separation time between job release instants. In the proposed Job Acceptance Multi-Server (JAMS), several CBSs share a common job queue, enabling flexible dispatching of a task's jobs to ready servers. The servers in JAMS incorporate an acceptance test, ensuring that only jobs with a guaranteed sufficient computation time prior to their deadline are dispatched. In an overload situation, this leads to JAMS dismissing jobs with unacceptable queue times.

The evaluation is performed for two types of workload: synthetically generated computation times from lognormal distributions, and computation times collected from a Model Predictive Control (MPC) task. Code and data are available[4]. The experiments show that JAMS successfully maintains the deadline miss rates of scheduled jobs below specified levels, at the cost of dismissing jobs during overload situations.

C5: An MAB application to improve average behavior while ensuring hard real-time guarantees.

C5 addresses research question **RQ5**, and is presented in Paper E. The task model is a Stochastic Parallel Synchronous Task, a special case of a DAG task

[3] https://github.com/annafriebe/ContMM_RT_BoundDMP

[4] https://github.com/annafriebe/RTAS_25_JAMS_AE

but a generalization of the Parallel Synchronous Task [21]. Each arm represents a choice of initial core allocation. The MAB is implemented as a bootstrap/ bag approximation [100] of Thompson sampling [33]. Response time bounds are derived for counterfactual reasoning. Given a response time observation with one initial core allocation, response time bounds are derived for other core allocations of the same job. In the proposed partial-feedback MAB, information about unexplored arms is derived from arms that have been explored, using the response time bounds.

In the evaluation, the proposed partial feedback MAB approach is compared to a Binary-Exponential Search approach presented in [94], a greedy method, and an MAB approach without the use of response time bounds. An energy model based on Dynamic Power Management (DPM) with sleep states for cores and CPU sockets is used to construct a reward function. Both MAB approaches reliably learn energy-efficient choices for core allocation, but using the response time bounds lowers the energy consumption.

6.3 Threats to Validity

In this section, threats to validity regarding the contributions are discussed.

When it comes to use case selection, there is a tradeoff between *conclusion* and *internal validity* on one hand and *external validity* on the other hand. Conclusion and internal validity are strengthened with a large variety of randomly configured task sets, while external validity is strengthened with realistic use cases. In the contributions of this thesis, a number of use cases are selected for the evaluation. In **C1**, **C3**, and **C4**, realistic use cases such as a video decompression task, a Furuta pendulum control task, and an MPC task are evaluated, and results from real systems are presented, increasing the external validity of these contributions. In **C4**, these results are complemented with results from a synthetic task, for greater conclusion and internal validity. In **C1** and **C2**, known Markov chain tasks are used in the evaluation, and **C2**, **C3**, and **C5** use simulation results in the evaluation. All these ensure that assumptions hold and increase conclusion validity. The results in **C2** and **C5** are from simulation only, which potentially is a threat to external validity. Therefore, generalization to more realistic use cases needs to be done with caution.

The data consistency approach to model validation used in **C1** produces a PFA_u measure – the probability of false alarm due to underdispersion. The cutoff value for when to reject an inconsistent model has not been determined in a systematic manner as outlined in [98]. This could be a threat to *construct* and *conclusion validity*. However, except for one macro state of the video decompression use case, all PFA_u values are based on several thousand observations. In such cases, a misspecified model should produce PFA_u values very close to 0 or 1.

In a few of the contributions, the baseline for comparison is relatively simple, or a comparison is performed between different versions of the same method, resulting in a threat to *construct validity* from a single intervention. The heuristic splitting step in the identification process of **C1** is also used in **C2** and **C3** and is not compared to alternative approaches. Although the results in the contributions confirm that the identification process results in models consistent with observations for the evaluated use cases, alternative splitting steps or identification procedures may produce even more consistent models. In **C2**, different versions of the adaptive procedure are compared, and in **C4**, a comparison is made with a simple baseline without dismissal. For both **C2** and **C4**, obvious alternatives for comparison have not been identified in the literature, but the lack of appropriate baselines makes it difficult to judge the performance of the contributions.

There are many potential use cases where the assumptions in **C2** do not hold, particularly the assumption that only emission distribution parameters change and not transition probabilities. This is a threat to the external validity of the approach.

Contributions **C3** and **C4** provide deductive proofs that ensure bounds on the deadline miss probability and dismissal probability, under the given assumptions. This implies that if we can ensure that the assumptions hold, then we know that the bounds hold irrespective of experimental results.

Providing publicly available code and data makes it easier for other researchers to build upon, reproduce results from, and find flaws in the contributions, strengthening the validity of research in general.

6.4 Included Papers

Paper A
Title: Identification and Validation of Markov Models with Continuous Emission Distributions for Execution Times.
Authors: Anna Friebe, Alessandro V. Papadopoulos, and Thomas Nolte.
Status: Published at IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2020.
Abstract: It has been shown that in some robotic applications, where the execution times cannot be assumed to be independent and identically distributed, a Markov Chain with discrete emission distributions can be an appropriate model. In this paper, we investigate whether execution times can be modeled as a Markov Chain with continuous Gaussian emission distributions. The main advantage of this approach is that the concept of distance is naturally incorporated. We propose a framework based on Hidden Markov Model (HMM) methods that 1) identifies the number of states in the Markov Model from observations and fits the Markov Model to observations, and 2) validates the proposed model with respect to observations. Specifically, we apply a tree-based cross-validation approach to automatically find a suitable number of states in the Markov model. The estimated models are validated against observations, using a data consistency approach based on log likelihood distributions under the proposed model. The framework is evaluated using two test cases executed on a Raspberry Pi Model 3B+ single-board computer running Arch Linux ARM patched with PREEMPT_RT. The first is a simple test program where execution times intentionally vary according to a Markov model, and the second is a video decompression using the `ffmpeg` program. The results show that in these cases, the framework identifies Markov Chains with Gaussian emission distributions that are valid models with respect to the observations.
Author's contributions: I was the main driver of this work, under the supervision of the co-authors. I have collected the timing traces, developed the code for the framework, and written a draft of the manuscript that was improved in collaboration with the co-authors.

Paper B
Title: Adaptive Runtime Estimate of Task Execution Times Using Bayesian Modeling.
Authors: Anna Friebe, Alessandro V. Papadopoulos, Filip Marković, and Thomas Nolte.
Status: Published at IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) 2021.
Abstract: In the recent works that analyzed execution-time variation of real-time tasks, it was shown that such variation may conform to regular behavior. This regularity may arise from multiple sources, e.g., due to periodic changes in hardware or program state, program structure, inter-task dependence, or inter-task interference. Such complexity can be better captured by a Markov Model, compared to the common approach of assuming independent and identically distributed random variables. However, despite the regularity that may be described with a Markov model, over time, the execution times may change due to irregular changes in input, hardware state, or program state. In this paper, we propose a Bayesian approach to adapt the emission distributions of the Markov Model at runtime, in order to account for such irregular variation. A preprocessing step determines the number of states and the transition matrix of the Markov Model from a portion of the execution time sequence. In the preprocessing step, segments of the execution time trace with similar properties are identified and combined into clusters. At runtime, the proposed method switches between these clusters based on a Generalized Likelihood Ratio (GLR). Using a Bayesian approach, clusters are updated, and emission distributions are estimated. New clusters can be identified, and clusters can be merged at runtime. The time complexity of the online step is O(N^2 + NC), where N is the number of states in the Hidden Markov Model (HMM) that is fixed after the preprocessing step, and C is the number of clusters.
Author's contributions: I have been the main author and driver of the work. Planning of the paper and evaluation has been performed with the co-authors. I have developed the software, performed the experiments, and written the draft of the paper that has been improved in collaboration with the co-authors.

Paper C

Title: Efficiently Bounding Deadline Miss Probabilities of Markov Chain Real-Time Tasks.

Authors: Anna Friebe, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte.

Status: Published in Real-Time Systems Special Issue: Resource Partitioning for Modern Multicore Systems in Oct 2024.

Abstract: In real-time systems analysis, probabilistic models, particularly Markov chains, have proven effective for tasks with dependent executions. This paper improves upon an approach utilizing Gaussian emission distributions within a Markov task execution model that analyzes bounds on deadline miss probabilities for tasks in a reservation-based server. Our method distinctly addresses the issue of runtime complexity, prevalent in existing methods, by employing a state merging technique. This not only maintains computational efficiency but also retains the accuracy of the deadline-miss probability estimations to a significant degree. The efficacy of this approach is demonstrated through the timing behavior analysis of a Kalman filter controlling a Furuta pendulum, comparing the derived deadline miss probability bounds against various benchmarks, including real-time Linux server metrics. Our results confirm that the proposed method effectively upper-bounds the actual deadline miss probabilities, showcasing a significant improvement in computational efficiency without significantly sacrificing accuracy.

Author's contributions: I am the main driver of the work. Planning of the paper has been performed jointly with the co-authors. I have provided the proofs, developed the software, performed the evaluation, and written the draft of the paper that has been improved in collaboration with the co-authors.

Paper D
Title: Nip It In the Bud: Job Acceptance Multi-Server.
Authors: Anna Friebe, Tommaso Cucinotta, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte.
Status: Accepted at 31st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) 2025.
Abstract: Computationally demanding tasks with highly variable execution times may require parallel processing. Scheduling such tasks with low deadline miss rates but without significant overprovisioning is challenging. This issue arises in applications like nonlinear optimization for Model Predictive Control (MPC). The Constant Bandwidth Server (CBS) provides timing isolation, supporting both hard and soft real-time tasks. However, scheduling parallel, time-varying jobs across multiple CBS instances requires static job-to-server assignments, which can lead to resource underutilization due to queued jobs awaiting specific servers. This paper introduces the Job Acceptance Multi-Server (JAMS), a mechanism in which multiple CBS instances share a common job queue, enabling flexible job dispatching for parallel workloads. JAMS incorporates a job dismissal mechanism to address overloads, ensuring that only jobs with guaranteed resource availability are accepted. Each CBS instance checks if it can complete a job by its deadline, given probabilistic knowledge on its execution times, dismissing unfeasible jobs to avoid excessive tardiness across queued tasks. Implemented in Linux, JAMS is evaluated with computation times drawn from an MPC task and synthetic datasets. The extensive experimental results we provide demonstrate that JAMS effectively controls the deadline miss rate, maintaining it below a specified design threshold.
Author's contributions: I am the main driver of the work. Planning of the paper has been performed jointly with the co-authors. I have provided the proofs, collected MPC traces for the evaluation, and written the draft of the paper that has been improved in collaboration with the co-authors. Tommaso Cucinotta has contributed the JAMS Linux implementation and performed most of the evaluation.

Paper E
Title: Resource Management For Stochastic Parallel Synchronous Tasks: Bandits To The Rescue.
Authors: Anna Friebe, Alberto Marchetti-Spaccamela, Tommaso Cucinotta, Alessandro V. Papadopoulos, Thomas Nolte, and Sanjoy Baruah.
Status: Under review.
Abstract: In scheduling real-time tasks, we face the challenge of meeting hard deadlines while optimizing for some other objective, such as minimizing energy consumption. Formulating the optimization as a Multi-Armed Bandit (MAB) problem allows us to use MAB strategies to balance the exploitation of good choices based on observed data with the exploration of potentially better options. In this paper, we integrate hard real-time constraints with MAB strategies for resource management of a Stochastic Parallel Synchronous Task. On a platform with M cores available for the task, $m \leq M$ cores are initially assigned. Prior work has shown how to compute a virtual deadline such that assigning all M cores to the task if it has not completed by this virtual deadline guarantees that the deadline will be met. An MAB strategy is used to select the value of m. A Dynamic Power Management (DPM) energy model considering CPU sockets and sleep states is described. Experimental evaluation shows that MAB strategies learn consistently suitable m, and perform well compared to binary exponential search and greedy methods.
Author's contributions: The idea to use a MAB formulation of this problem originated from Sanjoy Baruah. I am the main driver of the work. Planning of the paper has been performed jointly with the co-authors. I have developed the software, performed the evaluation, and written the draft of the paper that has been improved in collaboration with the co-authors. Alberto Marchetti-Spaccamela has contributed significantly to the final formulation of several of the proofs.

Chapter 7

Conclusion and Future Directions

7.1 Conclusion

In this thesis, real-time systems with highly varying computation times have been considered, and probabilistic methods for analysis and scheduling of such systems have been explored.

A first line of research explored periodic tasks, with dependence between successive job computation times. The first research question related to modeling the execution time behavior of such tasks. For evaluated use cases, it was shown that an Hidden Markov Model (HMM) with continuous emission distributions generates data consistent with observations. A framework was presented that identifies such an HMM from computation time traces and validates if the model output is consistent with observations using a dispersion-based statistic.

In the second research question, we consider such a task with runtime changes to emission distribution parameters. A method was presented for runtime adaptation of the HMM, assuming that emission distribution parameters change instantaneously but remain constant in segments between the change points.

The third research question regards the deadline miss probability of a

continuous-emission execution time HMM task in a reservation-based server. An efficient method was presented for bounding the expected deadline miss probability at steady state for such a task. The time required for the bound computation does not depend on the job execution time range, and there is no need to select a scaling factor. A state merging technique further reduces the bound computation time.

Further, we have explored the scheduling of parallel workload with varying computation times.

Research question four relates to ensuring a determined probability of meeting the deadline for started jobs. With the Job Acceptance Multi-Server (JAMS) mechanism, several Constand Bandwidth Servers (CBS) share a single job queue, enabling flexible job dispatching and implementing a guaranteed-capacity thread pool. Furthermore, the JAMS acceptance test ensures that started jobs have a specified probability of meeting the deadline, at the cost of dismissing jobs in overload situations.

In the fifth research question, we consider parallel synchronous tasks and optimize towards an average-case objective while meeting hard deadlines. Jobs of a stochastic parallel synchronous task scheduled on multiple processors are guaranteed to meet hard deadlines when assigned a maximum number of processors at a virtual deadline, based on previous work. A Multi-Armed Bandit (MAB) formulation of the problem enables learning an initial number of processors to assign to the task that minimizes the energy consumption. Using derived response time bounds in the MAB algorithm further decreases the energy consumption.

7.2 Future Directions

The validation procedure in Paper A performs an offline validation that data generated from the model is similar to empirical execution time traces, using a dispersion-based statistic. It would be interesting to pursue an online validation that the model is pessimistic in a stochastic sense compared to observed data.

The assumptions in the adaptive approach of Paper B are rather restrictive. In particular is assumed that only emission distribution parameters change, and that transition probabilities remain the same. There are many use cases where

this assumption does not hold. It would be interesting to explore adaptive approaches that allow for changes to the transition probabilities. Potentially, it could be feasible to consider a state-merged HMM as described in Paper C and estimate the highest-mean state, an upper bound on the remaining states, and transition probabilities to and from the highest-mean state.

For the JAMS in Paper D, there are several potential future directions to explore. The analysis could be provided for a general reservation-based server, not restricted to CBS. A natural step on the implementation side is to provide JAMS as a library with an interface for submitting work, checking status, and retrieving results. Extending JAMS with a dismissal probability estimate at the time of arrival would allow for the caller to manage overload situations in alternative ways. A kernel implementation would have some advantages in terms of overhead and could potentially allow for a tighter analysis by aligning task and server deadlines. Further experimental evaluation for different architectures and with bandwidth reclamation would be of interest.

In relation to the MAB approach in Paper E, Contextual Multi-Armed Bandit (CMAB) or restless bandit approaches could be explored to better adapt to a dynamic system. Validation of the energy model is necessary to ensure that the estimated energy savings translate to lower energy consumption in real life.

Seen in a wider perspective, analysis and scheduling of more complex real-time systems where the proposed contributions of this thesis constitute components is a natural future research direction. Distributed or edge/cloud systems could be considered, as well as analysis of more complex task models. In this thesis, the focus has been on computational requirements. Exploring the effects of memory access in relation to the contributions of this thesis is an important future direction.

Bibliography

[1] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *SIGCOMM Comput. Commun. Rev.*, 22(4):14–26, 1992.

[2] Pau Martí, Josep M Fuertes, Gerhard Fohler, and Krithi Ramamritham. Improving quality-of-control using flexible timing constraints: metric and scheduling. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 91–100, 2002.

[3] Giorgio C. Buttazzo and Marco Caccamo. Minimizing aperiodic response times in a firm real-time environment. *IEEE Transactions on Software Engineering*, 25(1):22–32, 1999.

[4] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem–overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36, 2008.

[5] Andreas Löfwenmark and Simin Nadjm-Tehrani. Fault and timing analysis in critical multi-core systems: A survey with an avionics perspective. *Journal of Systems Architecture*, 87:1–11, 2018.

[6] Reinhard Wilhelm. Mixed Feelings About Mixed Criticality (Invited Paper). In Florian Brandner, editor, *18th International Workshop on Worst-*

Case Execution Time Analysis (WCET 2018), volume 63 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:9, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[7] Alan Burns and Robert Ian Davis. Mixed criticality systems-a review. Technical report, Department of Computer Science, University of York, February 2022.

[8] Jaume Abella, Carles Hernandez, Eduardo Quiñones, Francisco J. Cazorla, Philippa Ryan Conmy, Mikel Azkarate-askasua, Jon Perez, Enrico Mezzetti, and Tullio Vardanega. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, 2015.

[9] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I Davis. An empirical survey-based study into industry practice in real-time systems. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 3–11. IEEE, 2020.

[10] Mario Günzel, Harun Teper, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. On the Equivalence of Maximum Reaction Time and Maximum Data Age for Cause-Effect Chains. In Alessandro V. Papadopoulos, editor, *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, volume 262 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:22, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[11] Guillem Bernat, Alan Burns, and Albert Liamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, 2001.

[12] T-S Tia, Zhong Deng, Mallikarjun Shankar, Matthew Storch, Jun Sun, L-C Wu, and JW-S Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings real-time technology and applications symposium (RTAS)*, pages 164–173. IEEE, 1995.

[13] Luca Abeni and Giorgio Buttazzo. QoS guarantee using probabilistic deadlines. In *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*, pages 242–249. IEEE, 1999.

[14] Mario Günzel, Niklas Ueter, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. Probabilistic reaction time analysis. *ACM Transactions on Embedded Computing Systems*, 22(5s):1–22, 2023.

[15] Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. Beyond the weakly hard model: Measuring the performance cost of deadline misses. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018.

[16] Nils Vreman, Anton Cervin, and Martina Maggio. Stability and performance analysis of control systems subject to bursts of deadline misses. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[17] Robert I Davis and Liliana Cucu-Grosjean. A survey of probabilistic timing analysis techniques for Real-Time systems. *Leibniz Transactions on Embedded Systems*, 6(1):03–1–03:60, May 2019.

[18] Robert Ian Davis and Liliana Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for Real-Time systems. *LITES: Leibniz Transactions on Embedded Systems*, pages 1–53, May 2019.

[19] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[20] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. A generalized parallel task model for recurrent real-time processes. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 63–72. IEEE, 2012.

[21] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems*, 49:404–435, 2013.

[22] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C. Buttazzo. Response-time analysis of conditional dag tasks in multiprocessor systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 211–221, 2015.

[23] José Carlos Fonseca, Vincent Nélis, Gurulingesh Raravi, and Luís Miguel Pinho. A multi-dag model for real-time parallel applications with conditional execution. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1925–1932, 2015.

[24] Alan Burns, Sanjoy K. Baruah, et al. Sustainability in real-time scheduling. *J. Comput. Sci. Eng.*, 2(1):74–97, 2008.

[25] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*, pages 4–13. IEEE, 1998.

[26] Alessandro Biondi, Alessandra Melani, and Marko Bertogna. Hard constant bandwidth server: Comprehensive formulation and critical scenarios. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pages 29–37. IEEE, 2014.

[27] Juri Lelli, Claudio Scordino, Luca Abeni, and Dario Faggioli. Deadline scheduling in the linux kernel. *Software: Practice and Experience*, 46(6):821–839, 2016.

[28] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[29] Jose Luis Diaz, Jose Maria Lopez, Manuel Garcia, Antonio M Campos, Kanghee Kim, and Lucia Lo Bello. Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *25th IEEE International Real-Time Systems Symposium*, pages 197–207. IEEE, 2004.

[30] Moshe Shaked. *Stochastic orders*. Springer series in statistics. Springer, New York, 2007.

[31] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. the IEEE*, 77(2):257–286, 1989.

[32] Peter Whittle. Restless bandits: Activity allocation in a changing world. *Journal of applied probability*, 25(A):287–298, 1988.

[33] Aleksandrs Slivkins. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[34] Alan Burns, Robert I Davis, Sanjoy Baruah, and Iain Bate. Robust mixed-criticality systems. *IEEE Transactions on Computers*, 67(10):1478–1491, 2018.

[35] Jian-Jia Chen, Mario Günzel, Peter Bella, Georg von der Brüggen, and Kuan-Hsun Chen. Dawn of the dead (line misses): Impact of job dismiss on the deadline miss rate. *arXiv preprint arXiv:2401.15503*, 2024.

[36] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, Katharina Morik, and Björn B Brandenburg. Efficiently approximating the worst-case deadline failure probability under EDF. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 214–226, 2021.

[37] Filip Marković, Pierre Roux, Sergey Bozhko, Alessandro V Papadopoulos, and Björn B Brandenburg. CTA: A correlation-tolerant analysis of the deadline-failure probability of dependent tasks. In *Proceedings of the 44th IEEE Real-Time Systems Symposium (RTSS)*, pages 317–330, 2023.

[38] Filip Marković, Georg von der Brüggen, Mario Günzel, Jian-Jia Chen, and Björn B Brandenburg. A distribution-agnostic and correlation-aware analysis of periodic tasks. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, pages 215–228. IEEE, 2024.

[39] José Luis Díaz, Daniel F García, Kanghee Kim, Chang-Gun Lee, L Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 289–300. IEEE, 2002.

[40] Liliana Cucu-Grosjean. Independence-a misunderstood property of and for probabilistic real-time systems. *In Real-Time Systems: the past, the present and the future*, pages 29–37, 2013.

[41] Robert I Davis, Alan Burns, and David Griffin. On the meaning of pWCET distributions and their use in schedulability analysis. In *In Proceedings Real-Time Scheduling Open Problems Seminar at (ECRTS'17)*, 2017.

[42] Sergey Bozhko, Filip Marković, Georg von der Brüggen, and Björn B Brandenburg. What Really is pWCET? A Rigorous Axiomatic Proposal. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 13–26, 2023.

[43] Kuan-Hsun Chen, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. Critical instant for probabilistic timing guarantees: Refuted and revisited. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 145–157. IEEE, 2022.

[44] Kevin Zagalo, Yasmina Abdeddaim, Avner Bar-Hen, and Liliana Cucu-Grosjean. Response time stochastic analysis for fixed-priority stable real-time systems. *IEEE Transactions on Computers*, 72(1):3–14, 2022.

[45] Filip Marković, Thomas Nolte, and Alessandro Vittorio Papadopoulos. Analytical approximations in probabilistic analysis of real-time systems. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 158–171, 2022.

[46] Guillem Bernat, Alan Burns, and Martin Newby. Probabilistic timing analysis: An approach using copulas. *Journal of Embedded Computing*, 1(2):179–194, 2005.

[47] Matthias Ivers and Rolf Ernst. Probabilistic network loads with dependencies and the effect on queue sojourn times. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 280–296. Springer, 2009.

[48] Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti,

Eduardo Quiñones, and Francisco J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *2012 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 91–101, 2012.

[49] George Lima, Dario Dias, and Edna Barros. Extreme value theory for estimating task execution time bounds: A careful look. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 200–211. IEEE, 2016.

[50] George Lima and Iain Bate. Valid application of EVT in timing analysis by randomising execution time measurements. In *23rd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17)*, pages 187–198. IEEE, 2017.

[51] Yue Lu, Thomas Nolte, Johan Kraft, and Christer Norström. Statistical-based response-time analysis of systems with execution dependencies between tasks. In *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 169–179, 2010.

[52] Yue Lu, Thomas Nolte, Johan Kraft, and Christer Norstrom. A statistical approach to response-time analysis of complex embedded real-time systems. In *2010 IEEE 16th international conference on embedded and real-time computing systems and applications*, pages 153–160. IEEE, 2010.

[53] Yue Lu, Thomas Nolte, Iain Bate, and Liliana Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 351–362. IEEE, 2012.

[54] Stuart Coles. *An Introduction to Statistical Modeling of Extreme Values*, volume 208. Springer-Verlag, London, UK, 2001.

[55] M Ross Leadbetter, Georg Lindgren, and Holger Rootzén. Conditions for the convergence in distribution of maxima of stationary normal processes. *Stochastic Processes and their Applications*, 8(2):131–139, 1978.

[56] Luca Santinelli, Jérôme Morio, Guillaume Dufour, and Damien Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In Heiko Falk, editor, *14th International Workshop on Worst-Case Execution Time Analysis*, volume 39 of *Open Access Series in Informatics (OASIcs)*, pages 21–30, Dagstuhl, Germany, 2014. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[57] Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Comput. Surv.*, 52(1), 2019.

[58] Alex F. Mills and James H. Anderson. A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. In *IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 207–217, 2011.

[59] Rui Liu, Alex F Mills, and James H Anderson. Independence thresholds: Balancing tractability and practicality in soft real-time stochastic analysis. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 314–323, 2014.

[60] Luca Abeni and Giorgio Buttazzo. Stochastic analysis of a reservation based system. In *Proceedings 15th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 946–952, 2001.

[61] Luigi Palopoli, Daniele Fontanelli, Luca Abeni, and Bernardo Villalba Frias. An analytical solution for probabilistic guarantees of reservation based soft real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):640–653, 2015.

[62] Luca Abeni, Daniele Fontanelli, Luigi Palopoli, and Bernardo Villalba Frías. A Markovian model for the computation time of real-time applications. In *2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–6. IEEE, 2017.

[63] Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. Probabilistic real-time guarantees: There is life beyond the

iid assumption (outstanding paper). In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 175–186. IEEE, 2017.

[64] Sorin Manolache, Petru Eles, and Zebo Peng. Schedulability analysis of applications with stochastic task execution times. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(4):706–735, 2004.

[65] Paolo Pazzaglia, Claudio Mandrioli, Martina Maggio, and Anton Cervin. DMAC: Deadline-Miss-Aware Control. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:24, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[66] Sanjoy K Baruah, Alan Burns, and Robert I Davis. Response-time analysis for mixed criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 34–43. IEEE, 2011.

[67] Huang-Ming Huang, Christopher Gill, and Chenyang Lu. Implementation and evaluation of mixed-criticality scheduling approaches for sporadic tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(4s):1–25, 2014.

[68] Iain Bate, Alan Burns, and Robert I Davis. An enhanced bailout protocol for mixed criticality embedded software. *IEEE Transactions on Software Engineering*, 43(4):298–320, 2016.

[69] Angeliki Kritikakou, Claire Pagetti, Olivier Baldellon, Matthieu Roy, and Christine Rochange. Run-time control to increase task parallelism in mixed-critical systems. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 119–128. IEEE, 2014.

[70] Marco Spuri, Giorgio Buttazzo, and Fabrizio Sensini. Robust aperiodic scheduling under dynamic priority systems. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 210–219. IEEE, 1995.

[71] Luca Abeni and Giorgio Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27:123–167, 2004.

[72] Zelin Tong, Shareef Ahmed, and James H Anderson. Holistically budgeting processing graphs. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 27–39. IEEE, 2023.

[73] Amy R Ward. Asymptotic analysis of queueing systems with reneging: A survey of results for FIFO, single class models. *Surveys in Operations Research and Management Science*, 17(1):1–14, 2012.

[74] Łukasz Kruk, John Lehoczky, Kavita Ramanan, and Steven Shreve. Heavy traffic analysis for EDF queues with reneging. *The Annals of Applied Probability*, 21(2):484–545, 2011.

[75] Don Towsley and S. S. Panwar. Optimality of the stochastic earliest deadline policy for the g/m/c queue serving customers with deadlines. Technical report, USA, 1991.

[76] Ward Whitt. Fluid models for multiserver queues with abandonments. *Operations research*, 54(1):37–54, 2006.

[77] Ward Whitt. Stabilizing performance in a single-server queue with time-varying arrival rate. *Queueing Systems*, 81:341–378, 2015.

[78] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 1(4):397–413, 1993.

[79] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.

[80] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):1–34, 2016.

[81] Saad Zia Sheikh and Muhammad Adeel Pasha. Energy-efficient multicore scheduling for hard real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(6):1–26, 2018.

[82] Guoqi Xie, Xiongren Xiao, Hao Peng, Renfa Li, and Keqin Li. A survey of low-energy parallel scheduling algorithms. *IEEE Transactions on Sustainable Computing*, 7(1):27–46, 2021.

[83] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: the laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, page 1–8, USA, 2010. USENIX Association.

[84] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, 35(5):89–102, October 2001.

[85] Hakan Aydin, Rami Melhem, Daniel Mosse, and Pedro Mejia-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings 13th Euromicro Conference on Real-Time Systems*, pages 225–232. IEEE, 2001.

[86] Agostino Mascitti, Tommaso Cucinotta, Mauro Marinoni, and Luca Abeni. Dynamic partitioned scheduling of real-time tasks on arm big.little architectures. *Journal of Systems and Software*, 173:110886, 2021.

[87] Luca Abeni and Tommaso Cucinotta. Adaptive partitioning of real-time tasks on multiple processors. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, page 572–579, New York, NY, USA, 2020. Association for Computing Machinery.

[88] Agostino Mascitti and Tommaso Cucinotta. Dynamic partitioned scheduling of real-time dag tasks on arm big.little architectures. In *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, RTNS '21, page 1–11, New York, NY, USA, 2021. Association for Computing Machinery.

[89] Robert Schöne, Daniel Molka, and Michael Werner. Wake-up latencies for processor idle states on current x86 processors. *Computer Science-Research and Development*, 30:219–227, 2015.

[90] Georgia Antoniou, Davide Bartolini, Haris Volos, Marios Kleanthous, Zhe Wang, Kleovoulos Kalaitzidis, Tom Rollet, Ziwei Li, Onur Mutlu, Yiannakis Sazeides, and Jawad Haj Yahya. Agile C-states: A core C-state architecture for latency critical applications optimizing both transition and cold-start latency. *ACM Trans. Archit. Code Optim.*, 21(4), November 2024.

[91] Xin Zhan, Reza Azimi, Svilen Kanev, David Brooks, and Sherief Reda. Carb: A c-state power management arbiter for latency-critical workloads. *IEEE Computer Architecture Letters*, 16(1):6–9, 2016.

[92] Erfan Sharafzadeh, Seyed Alireza Sanaee Kohroudi, Esmail Asyabi, and Mohsen Sharifi. Yawn: A CPU idle-state governor for datacenter applications. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 91–98, 2019.

[93] Chih-Hsun Chou, Laxmi N. Bhuyan, and Daniel Wong. μdpm: Dynamic power management for the microsecond era. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 120–132. IEEE, 2019.

[94] Alessandro V. Papadopoulos, Kunal Agrawal, Enrico Bini, and Sanjoy Baruah. Feedback-based resource management for multi-threaded applications. *Real-Time Systems*, pages 1–34, 2022.

[95] Gordana Dodig-Crnkovic. Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden*, pages 126–130. sn, 2002.

[96] Claes. Wohlin, Per. Runeson, Martin. Höst, Magnus C. Ohlsson, Bjö. Regnell, and Anders. Wesslén. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2nd ed. 2024. edition, 2024.

[97] Takahiro Shinozaki. HMM state clustering based on efficient cross-validation. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages 1157–1160, 2006.

[98] Andreas Lindholm, Dave Zachariah, Peter Stoica, and Thomas B Schön. Data consistency approach to model validation. *IEEE Access*, 7:59788–59796, 2019.

[99] Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. The prosit tool: Toward the optimal design of probabilistic soft real-time systems. *Software: Practice and Experience*, 48(11):1940–1967, 2018.

[100] Nikunj C. Oza and Stuart J. Russell. Online bagging and boosting. In Thomas S. Richardson and Tommi S. Jaakkola, editors, *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, volume R3 of *Proceedings of Machine Learning Research*, pages 229–236. PMLR, 04–07 Jan 2001. Reissued by PMLR on 31 March 2021.

II

Included Papers

Chapter 8

Paper A
Identification and Validation of Markov Models with Continuous Emission Distributions for Execution Times.

Anna Friebe, Alessandro V. Papadopoulos, and Thomas Nolte.
In IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2020.

Abstract

It has been shown that in some robotic applications, where the execution times cannot be assumed to be independent and identically distributed, a Markov Chain with discrete emission distributions can be an appropriate model. In this paper we investigate whether execution times can be modeled as a Markov Chain with continuous Gaussian emission distributions. The main advantage of this approach is that the concept of distance is naturally incorporated. We propose a framework based on Hidden Markov Model (HMM) methods that 1) identifies the number of states in the Markov Model from observations and fits the Markov Model to observations, and 2) validates the proposed model with respect to observations. Specifically, we apply a tree-based cross-validation approach to automatically find a suitable number of states in the Markov model. The estimated models are validated against observations, using a data consistency approach based on log likelihood distributions under the proposed model. The framework is evaluated using two test cases executed on a Raspberry Pi Model 3B+ single-board computer running Arch Linux ARM patched with PREEMPT_RT. The first is a simple test program where execution times intentionally vary according to a Markov model, and the second is a video decompression using the `ffmpeg` program. The results show that in these cases the framework identifies Markov Chains with Gaussian emission distributions that are valid models with respect to the observations.

8.1 Introduction

In real-time systems requirements on timing properties must be considered, in addition to functional requirements, i.e., it is of importance to have the correct behavior *at the appropriate time*. Real-time requirements range from safety critical timing requirements of *hard real-time systems* found in applications of aeronautics, automotive and medical device systems to *soft real-time systems*, e.g., common in multimedia applications. Failure to meet hard real-time requirements may result in a disaster and/or loss of human life whereas failure to meet a soft real-time requirement can cause a deterioration of the Quality of Service (QoS) [11] such as in a video playback or affect the Quality of Control (QoC) [34] of a robot's motion planning.

It is a challenge to enable sufficiently accurate timing analysis of today's complex systems. Multicore processors [28] and mixed-criticality systems [7, 43], as well as fog and edge computing capabilities [41, 40, 12], require new methods for ensuring sound timing estimates along with functional integrity and limited over-provisioning of computational resources and bandwidth for communication. Practical timing analysis methods that consider the entire timing distribution, as opposed to only the tail of the distribution, can allow for system design without excessive over-provisioning. Taking the entire distribution into account is of particular interest primarily in the case of soft real-time applications where requirements on QoS or QoC are considered.

Frías *et al.* have shown that computation times of a computer vision application in a robotic system can be described as a Markov Model [20, 4]. Inspired by the work of Frías *et al.*, in this paper we investigate the following research question: How can the execution time distribution of a task be faithfully modeled in a probabilistic framework? In particular:

1. Can execution time distributions be suitably modeled as a Markov Chain, where each state is associated with a Gaussian emission distribution?

2. How can one estimate the model parameters from timing measurements of the task's jobs?

Our main motivation for exploring *continuous emission distributions* is that they naturally include a concept of distance. Two execution time measure-

ments that are similar are more likely to originate from the same state, compared to measurements of different magnitude. In the standard methods for Markov models with discrete emission distributions, each execution time value is treated as a label and the distance information is lost. By using continuous distributions, a model likely to provide a reasonable estimate from a smaller amount of observations. Although each execution time is a discrete number of clock cycles, realistic tasks on today's processors often result in a large number of possible values, that can be closely approximated by a continuous distribution. In order to develop methods that can be of practical use in schedulability analysis, we estimate model parameters from observations.

In this paper, we present an automated framework that estimates and validates an execution time distribution model from observations. The proposed model is a Markov Model with a Gaussian emission distribution associated with each state. More precisely, we propose an HMM, as we observe the execution times, but the states cannot be directly observed. Firstly, in **step 1** we identify the number of states for the HMM, and fit the model to the observations. A *tree-based cross-validation approach* [39] is adopted for identifying the number of states. We estimate the parameters for the Gaussian distributions and the transition matrix by applying the *Expectation-Maximization algorithm* [35], initialized with values resulting from the tree-based cross-validation. In **step 2** we validate the estimated model using observations. Here, we adopt a *data consistency approach* [27], and derive methods for application of this approach to Markov Chains using outputs from the *Forward-backward algorithm*.

A set of probabilistic techniques are selected and combined in the framework, to enable identification and validation of the HMM. The methods are applied to two test cases, a test program with a known Markov Chain behavior, and a video decompression program treated as a black box. The results are presented and discussed. Further investigation is needed to evaluate the applications where the framework and the specific techniques of each step are most suitable, and for what applications other techniques are better for one or several of the proposed steps.

The rest of the paper is structured as follows. Section 8.2 presents the related work, followed by Section 8.3 that presents the task model. Section 8.4 presents the proposed framework. Sections 8.5 and 8.6 discuss the experimental

results, Finally, Section 8.7 concludes the paper and highlight directions for future work.

8.2 Related Work

Cucu-Grosjean and Davis have recently provided thorough surveys of the literature on probabilistic methods in Timing Analysis [15], Response Time Analysis, analysis of server-based systems, Real-Time Queuing Theory, system analysis with fault modeling and Mixed Criticality Systems [14]. Cazorla *et al.* provide a taxonomy and a survey on the methods used in Probabilistic Worst-Case Execution Time Analysis [9]. The authors also emphasize the fact that while measurement-based approaches may allow analysis of a black-box system, the results are only reliable if the analysis data are representative with respect to the operational environment. That is, all sources of variation in execution times or latencies that are relevant for the result need to be contributing to the variation in the data. Otherwise their effects need to be upper-bounded or accounted for in other ways.

In the area of static probabilistic timing analysis, many works consider models for set-associative or fully associative caches. Quinones *et al.* [37] showed that for some cases with programs displaying a cache risk pattern, random replacement gives better results and lower variability compared to Least Recently Used (LRU) replacement. Altmeyer *et al.* [6] provide analysis considering reuse distance, associativity and contention. Analysis using random replacement caches has been extended to the multi-path case [26, 25]. Chen and Beltrame [10] perform timing analysis for single-path programs on systems with evict-on-miss random replacement caches by using an adaptive Markov model.

Measurement-Based Probabilistic Timing Analysis methods estimate the pWCET by applying statistical techniques to observations of execution time measurements. While WCET is a scalar value – the upper bound of the worst case execution time of runs – the pWCET is a probability distribution representing the upper bound on the probability of exceeding each execution time value in valid scenarios of repeated runs of the program. The theoretical basis is in Extreme Value Theory (EVT). The first work in this direction was by

Burns, Edgar and Griffin [8, 19, 21]. Measurement-Based Probabilistic Timing Analysis was then introduced by Cucu-Grosjean *et al.* [13] in 2012.

Probabilistic Response Time Analysis is used to calculate the response time distribution of jobs, and in this manner estimate the probability of a deadline miss. Diaz *et al.* [16] presented response time analysis for a system with periodic tasks where random variables describe execution times. Here, the worst-case processor utilisation can exceed 1, since a backlog is considered at the end of the hyperperiod. They show that the backlog is a Markov chain. In [17] they also provided properties needed to achieve a safe over-approximation. More recent, the system model was extended by Kaczynski *et al.* [22] to also allow for systems with aperiodic tasks.

Similarly as in Measurement-Based Probabilistic Timing Analysis for pWCET estimates, EVT has also been applied in order to estimate response time distributions. The majority of the work in this line of research, Statistical Response Time Analysis, has been performed by Lu *et al.* [32, 31, 29, 30].

Real-Time Queueing Theory is an area where queue lengths and waiting times are analyzed mathematically. Lehozcky [24] introduced the concept in 1996, building upon work on queuing theory that started in the 1950s. Doytchinov provided a mathematical formalization in [18].

Probabilistic analysis has also been applied in analysis of server-based systems. Buttazzo and Abeni introduced the Constant Bandwidth Server (CBS) [1], and probabilistic deadlines for Quality of Service guarantees [2]. The same group has considered execution times [3, 36] and interarrival times [5, 33, 36] modeled with probability distributions.

Frías *et al.* [20, 4] have published work regarding execution time models for tasks where execution times display dependencies due to slowly changing input data. They have shown that for a robotic image processing task in line following, modeling execution times as a Hidden Markov Model is appropriate. In this work, discrete emission distributions for the different states are used. The deadline miss probability under CBS/Earliest Deadline First (EDF) is estimated for the Hidden Markov Model and compared to an assumption of independent and identically distributed random variables. The calculated probabilities are compared to experimental results with CBS/EDF as implemented in the Linux SCHED_DEADLINE scheduling policy. The experiments show that with an

independent and identically distributed (i.i.d.) assumption of execution times, the probability of respecting the deadline is overestimated, i.e., the estimate is optimistic. The estimates based on the identified Hidden Markov Model, on the other hand, are very close to the experimental results.

8.3 Task Model

In this paper, we consider a periodic task τ consisting of a sequence of periodic jobs J_i, $i \in \mathbb{N}$, with period T. Each job has an execution time $c_i \in \mathbb{R}$.

We model the execution time distribution of the task according to an adapted version of the Markov Computation Time Model (MCTM) in Frías *et al.* [20]. The model is described by the set $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$, where

- $\mathcal{M} = \{m_1, m_2, \ldots, m_N\}$ is the set of N states, $m_n, n \in \mathbb{N}$.
- $\mathcal{P}$ is the $N \times N$ state transition matrix, where the element $p_{a,b}$ represents the conditional probability $\mathbb{P}(X_{i+1} = m_b | X_i = m_a)$ of being in state m_b at round $i+1$, given that at round i the state is m_a.
- $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$ is the set of execution time distributions, or emission distributions related to respective state. In this paper, these are modelled as Gaussian distributions with mean μ_n, and variance σ_n^2, i.e., $C_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$.

8.4 Framework

In this section, we present and describe the framework that we have developed for the identification and validation of the probabilistic model. Specifically, the framework consists of the following steps:

1. Firstly, we apply the tree-based cross-validation approach [39] described in Section 8.4.1 to identify the number of states in the HMM from the observations. An HMM with the identified number of states is fitted to

the observations, according to the Expectation-Maximization [35] algorithm, using the likelihoods obtained with the Forward-backward algorithm [38]. The Gaussian distribution parameters and the transition matrix used as a starting point for the optimization is given by the outputs of the tree-based cross-validation.

2. Secondly, the obtained model is validated using a data consistency approach [27] described in Section 8.4.2. Here we derive expressions using outputs from the Forward-backward algorithm for application of the data consistency model validation.

In the following subsections we describe these steps on model identification and validation in more detail.

8.4.1 Tree-Based Cross-Validation

In general, the number of states N is not known a priori, and must be identified, for example based on logged data. In order to identify a number of states N_{opt} that allows for capturing the execution sequence properties without overfitting, a tree-based cross-validation approach is applied, as described in Shinozaki [39]. The execution time sequence $cs = \{c_1, c_2, \ldots, c_{NS}\}$ consisting of execution times from $NS \in \mathbb{N}$ jobs, is split into M non-overlapping folds cs_f with index f.

$$cs = \cup_{f=1}^{M} cs_f$$
$$cs_f \cap cs_g = \emptyset, \quad f \neq g$$

For each fold with index f, we also define the complement cs_f^c, the remaining folds:

$$cs_f^c = \cup_{f \neq g} cs_g$$

For each fold an MCTM with $N > N_{\text{opt}}$ states is fitted to the remaining folds cs_f^c. The initial values of means and standard deviations for the emission distributions are given by k-means clustering with $k = N$.

The occupancy probability γ_{ni} is the probability of being in state n at round i, given the observation sequence cs_f, where $c_i \in cs_f$ and the model parameters

retrieved from fitting to cs_f^c. The occupancy probabilities for each state index n and observation round i are calculated with the Viterbi algorithm [38].

Statistics containing all information from the sample needed for parameter value estimates for a statistical model are sufficient for the parameter. By calculating the sufficient statistics we can store the needed information from a sample in a compact manner. Sufficient statistics for likelihood estimates of a Markov chain model with Gaussian emission distribution are $a0$, $a1$ and $a2$ [39]. These are calculated for each fold index f and state index n:

$$a0_{fn} = \sum_{i,c_i \in cs_f} \gamma_{ni}$$
$$a1_{fn} = \sum_{i,c_i \in cs_f} c_i \gamma_{ni}$$
$$a2_{fn} = \sum_{i,c_i \in cs_f} c_i^2 \gamma_{ni}$$

For a set or cluster s of states, the maximum likelihood mean μ and variance ν for a fold index f can be calculated from the sufficient statistics from remaining folds:

$$\mu_{fs} = \frac{\sum_{g \neq f} \sum_{m_n \in s} a1_{gn}}{\sum_{g \neq f} \sum_{m_n \in s} a0_{gn}} \tag{8.1}$$

$$\nu_{fs} = \frac{\sum_{g \neq f} \sum_{m_n \in s} a2_{gn}}{\sum_{g \neq f} \sum_{m_n \in s} a0_{gn}} - \mu_{fs}^2 \tag{8.2}$$

These are then used to calculate a likelihood per fold index f and cluster s:

$$L_{fs} = -\frac{1}{2} \times \sum_{m_n \in s} \left(\ln(2\pi\nu_{fs}) a0_{fn} + \frac{a2_{fn} - 2\mu_{fs} a1_{fn} + \mu_{fs}^2 a0_{fn}}{\nu_{fs}} \right) \tag{8.3}$$

The likelihood for a cluster is then calculated by summation of the likelihoods of each fold index.

$$L_s = \sum_{f=1}^{M} L_{fs} \tag{8.4}$$

A tree is created, and initially all states are placed in a cluster in the root node. The cross validated likelihood is calculated for the tree consisting of only this cluster. Attempts are made to split the leaf nodes of the tree, so that the node's cluster is split into two clusters. The possible ways of splitting the states in a tree node are:

1. 2-means clustering of 2D data points of mean and standard deviation of each state is performed, and the resulting split is evaluated.

2. The states are ordered with respect to increasing mean, and each possible split along the ordered states is evaluated.

3. The modes are ordered with respect to increasing standard deviation, and each possible split along the ordered states is evaluated.

The split that gives the greatest increase in likelihood is selected. Nodes are split as long as the cross validated likelihood of the subtree is increased by the split, or until there is only one state in the tree node. The node splitting process is a greedy algorithm that may lead to a local maximum. Pseudo code is provided in Algorithm 8.1.

When a suitable number of states has been found, an MCTM with this number of states is fitted to the execution time samples, starting from initial values taken from the node clusters.

8.4.2 Data Consistency Model Validation

We evaluate whether the fitted model as described in Section 8.4.1,is valid, with respect to observations. Thus, we generate samples from the model and using a data consistency approach [27] we compare the generated samples to observations. If the observed data is consistent with data generated from the model, the model can be used in schedulability analysis.

The model validation can be performed with the same observations used for the model estimate, to evaluate whether the model can capture the properties of the observations. The evaluation can also be performed with observations from other runs of the program, to evaluate whether the model and parameters are valid in these cases, for different inputs or different hardware states.

Algorithm 8.1: Pseudo code describing the tree cluster splitting process. The likelihood increase for possible splits of a cluster are calculated using the pre-computed sufficient statistics.

Input: $suffStats$, N
Output: $tree$

Function `TreeClusterSplitting(`$suffStats$, N`)`**:**
  $tree \leftarrow createNode()$
  `/* Add all states to the root cluster */`
  $tree.states \leftarrow [1:N]$
  `/* Find the best split */`
  $tree.[leftStates, rightStates, advantage] \leftarrow$ `CalcSplitAdvantage(`$tree, suffStats$`)`
  `/* While the likelihood increases, and we can split leaves */`
  while $(tree.advantage > 0)$ ***and*** $(nLeafNodes(tree) \leq N)$ **do**
    for $node \in leaves(tree)$ **do**
      if $node.advantage > 0$ **then**
        `/* Add new leaf nodes and split the state cluster */`
        $node.leftChild \leftarrow$ `CreateNode()`
        $node.leftChild.states \leftarrow node.leftStates$
        $node.rightChild \leftarrow$ `CreateNode()`
        $node.rightChild.states \leftarrow node.rightStates$
        $node.states \leftarrow \emptyset$
    for $node \in leaves(tree)$ **do**
      `/* Find the best split of the leaf */`
      $node.[leftStates, rightStates, advantage] \leftarrow$ `CalcSplitAdvantage(`$node, suffStats$`)`
    for $node \in tree; post-order$ **do**
      `/* Move the highest likelihood increase to the root */`
      $node.advantage \leftarrow$ `MaxAdvantageChildren(`$node$`)`
  `/* Return the tree with` N_{opt} `leaf clusters */`
  return $tree$

The data consistency approach we apply is described by Lindholm *et al.* [27]. The log-likelihood under the proposed model is estimated for samples generated from the model and for the observed samples. Using a log-likelihood based statistic, an estimate is calculated of the probability of generating the observed sample or a sample with higher dispersion, from the evaluated model. This is equivalent to the probability that we reject the model on the basis of the observed data being overdispersed, assuming that the data is generated from the model. Another way of describing it is that the model is underdispersed compared with the observations. This probability of falsely rejecting the model or Probability of False Alarm due to underdispersion (PFA_u) is similar to the p-value concept in hypothesis evaluation. While the p-value is the probability of obtaining the test results or more extreme values assuming the null hypothesis is correct, the PFA_u is the probability of obtaining data with at least the observed variability, assuming they are generated from the proposed model.

We denote the observed execution times with an underline, as $\underline{\mathbf{c}}$. In our case, we evaluate a single model with a probability distribution $p(\mathbf{c}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$, where $\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*$ are the parameters of the fitted MCTM.

Using the model, we generate trajectories denoted with tilde $\tilde{\mathbf{c}} \sim p(\mathbf{c}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$. Using $c_{1:t}$ to denote the samples at rounds 1 to t from the trajectory, the conditional likelihood of an execution time measurement in a trajectory under the model is:

$$p_t = p(c_t|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = \frac{p(c_{1:t}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)}{p(c_{1:t-1}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)}$$

This can be calculated from the scaling factors resulting from the Forward-backward algorithm. We denote these as sca_i. From Rabiner [38] we have the probability of the observations expressed in terms of the scaling factors:

$$p(c_{1:t}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = \frac{1}{\prod_{i=1}^{t} sca_i}$$

From this it is clear that the conditional probability can be written as:

$$p_t = p(c_t|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = \frac{1}{sca_t}$$

The conditional log-likelihood of a data point is:

$$z_t \triangleq \ln p(c_t|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = -\ln sca_t \tag{8.5}$$

Conditional probabilities of outputs for each state separately can be estimated using the transition matrix and the scaled forward variables $\hat{\alpha}$:

$$\begin{aligned}
&p(c_t, X_t = j|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) \\
&= p(c_t|X_t = j) \sum_{k=1}^{N} p_{k,j} p(X_{t-1} = k|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) \\
&= \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{(c_t-\mu_j)^2}{2\sigma^2}} \sum_{k=1}^{N} p_{k,j} \hat{\alpha}_{k,t-1}
\end{aligned}$$

From Rabiner [38] we have that:

$$\begin{aligned}
\alpha_{k,t} &= p(c_{1:t}, X_t = k|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) \\
\hat{\alpha}_{k,t} &= \alpha_{k,t} \prod_{i=1}^{t} sca_i = \frac{p(c_{1:t}, X_t = k|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)}{p(c_{1:t}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)} \\
&= p(X_t = k|c_{1:t}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)
\end{aligned}$$

The conditional log-likelihood of each state and data point is:

$$\begin{aligned}
&z_{t,j} \triangleq \ln p(c_t|X_t = j, c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) \\
&= -\ln \sigma_j - \frac{\ln 2\pi}{2} - \frac{(c_t - \mu_j)^2}{2\sigma^2} + \ln \sum_{k=1}^{N} p_{k,j} \hat{\alpha}_{k,t-1}
\end{aligned} \tag{8.6}$$

We denote the mean of the log-likelihood of data points in generated trajectories as $\mathbb{E}[\tilde{z}_t]$ and the variance as $\mathbf{Var}[\tilde{z}_t]$. The test statistic T for a trajectory is defined:

$$T(\mathbf{c}; \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = \frac{1}{n} \sum_{t=1}^{n} \frac{z_t - \mathbb{E}[\tilde{z}_t]}{\mathbf{Var}[\tilde{z}_t]} \tag{8.7}$$

T statistics are defined similarly for each state by replacing z_t with $z_{t,j}$. S is defined as the random event of a generated sample resulting in a higher T-statistic than the observed one:

$$S(\tilde{\mathbf{c}}, \underline{\mathbf{c}}) : T(\tilde{\mathbf{c}}; \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) > T(\underline{\mathbf{c}}; \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$$

When the probability of S, $\mathbb{P}_{\tilde{\mathbf{c}}|\mathcal{M}_*,\mathcal{P}_*,\mathcal{C}_*}(S(\tilde{\mathbf{c}}, \underline{\mathbf{c}}))$ is close to 0 or close to 1, it indicates that the observed data is inconsistent with the proposed model.

Lindholm *et al.* define PFA_u, the probability of falsely rejecting a model due to under-dispersion of the generated log likelihoods as:

$$PFA_u \triangleq \mathbb{P}_{\tilde{\mathbf{c}}|\mathcal{M}_*,\mathcal{P}_*,\mathcal{C}_*}(S(\tilde{\mathbf{c}}, \underline{\mathbf{c}})) \tag{8.8}$$

Lindholm *et al.* also define the probability of falsely rejecting the model due to under- or overdispersion as:

$$PFA = \min(PFA_u, 1 - PFA_u)$$

However, in our work, we use PFA_u, as an under-dispersion of data generated from the proposed model indicates that the model is optimistic with regard to tail estimates. We note that values of PFA_u that are close to 1 also indicate model inconsistency, but that this relates to over-dispersion of data generated from the model.

Pseudo code for the data consistency approach is given in Algorithm 8.2.

8.5 Evaluation

8.5.1 Test Setup

A Raspberry Pi 3B+ single board computer with quad-core 1.4 GHz BCM2837B0 is utilized in the tests. Arch Linux ARM kernel 4.14.87 with `PREEMPT_RT` patch 4.14.87-49 is configured with fully preemptible kernel and timer frequency of 100Hz. The SD card low latency mode and `dwc_otg` FIQ are disabled. A test program is pinned to a core that is isolated from load-balancing and scheduling algorithms. The scaling governor is set to performance for all cores and USB is disabled during the run. The program is

run in user space with FIFO scheduling and maximum priority. The `ftrace` utility `trace-cmd` is used to log release (`sched_wakeup`) and scheduling (`sched_switch`) events, and to generate trace reports with nanosecond precision from the trace logs.

The model identification and validation framework is applied offline using the recorded traces.

8.5.2 Implementation

The tree-based cross validation approach described in Section 8.4.1 and the data consistency criterion model validation described in Section 8.4.2 are implemented in R[1], utilizing the R packages `depmixS4` [42] and `data.tree`. Evaluation code as well as test programs and scripts are available online [2].

Four folds are used for the cross validation. As described in Section 8.4.1, the MCTM is fitted to three folds, and the sufficient statistics using the fitted model are calculated for the remaining fold. The occupancy probabilities are determined by application of the Viterbi algorithm. A new MCTM with the number of states given by the tree-based cross validation is created, and initialized with the means and variances from the clusters, as given in Eqs. (8.1) and (8.2) averaged over all folds. This model is then fitted to the entire training set, and the fitted model is validated with the data consistency criterion.

Values of the probability of false alarm due to underdispersion, PFA_u, are estimated for the entire model using z_t as in Eq. (8.5), and for each state in the model using $z_{t,j}$ as in Eq. (8.6). First, 100 trajectories are generated for estimation of the mean and variance of the log likelihood. The trajectories are generated using the simulate function in `depmixS4`, and log likelihoods are retrieved from `depmixS4`'s forward and scaling variable resulting from the Forward-backward algorithm. Second, 100 new trajectories are generated for calculation of T values as given by Eq. (8.7). Referring to Algorithm 8.2, both M' and M are set to 100.

[1] https://www.r-project.org/

[2] https://github.com/annafriebe/MarkovChainETFramework

8.5.3 Markov Chain Test Program

In a first test, a program with a known Markov Chain behavior is evaluated. The test program contains a state machine with three states. The program keeps an array of 100 integers, initialized from a random uniform distribution in the range [0, 4711]. It executes a job periodically at a 5ms interval. In the job, a state transition is performed, given the following transition matrix:

$$\mathcal{P} = \begin{pmatrix} 0.7 & 0.1 & 0.2 \\ 0.5 & 0.1 & 0.4 \\ 0.5 & 0.2 & 0.3 \end{pmatrix} \tag{8.9}$$

Depending on the current state, elements in the array are increased with 43 and a modulo operation with 4711 is performed. The first state has the shortest average execution time, the second state the middle and the third state the longest average execution time.

Logs are created from 21 runs of the program, one is used for model parameter estimation, and 20 in the model evaluation. In each run, the task releases 10 000 jobs. A python script is used to calculate the execution time for each job. The steady state is considered, so the logs from the first 250 jobs and the last executed job instance are excluded. The execution times of the first 250 are slightly lower, due to the program always starting in state 1 and possibly due to the system state. The last job's execution time is much longer due to produced status output before termination.

The execution time sequence used for estimating models is displayed in Fig. 8.1.

The tree based cross-validation approach is applied with 8 initial states to the training execution time sequence. The fitted model has six remaining states. The means and standard deviations of the states and PFA_u values are displayed in Table 8.1, and the estimated transition matrix is given by:

$$\mathcal{P} = \begin{pmatrix} 0.51 & 0.18 & 0.08 & 0.02 & 0.19 & 0.007 \\ 0.45 & 0.27 & 0.05 & 0.04 & 0.18 & 0.005 \\ 0.36 & 0.14 & 0.07 & 0.047 & 0.38 & 0.005 \\ 0.31 & 0.18 & 0.07 & 3.7 \times 10^{-5} & 0.43 & 0.012 \\ 0.34 & 0.15 & 0.15 & 0.06 & 0.30 & 0.008 \\ 0.12 & 0.45 & 0.02 & 0.18 & 0.12 & 0.11 \end{pmatrix} \tag{8.10}$$

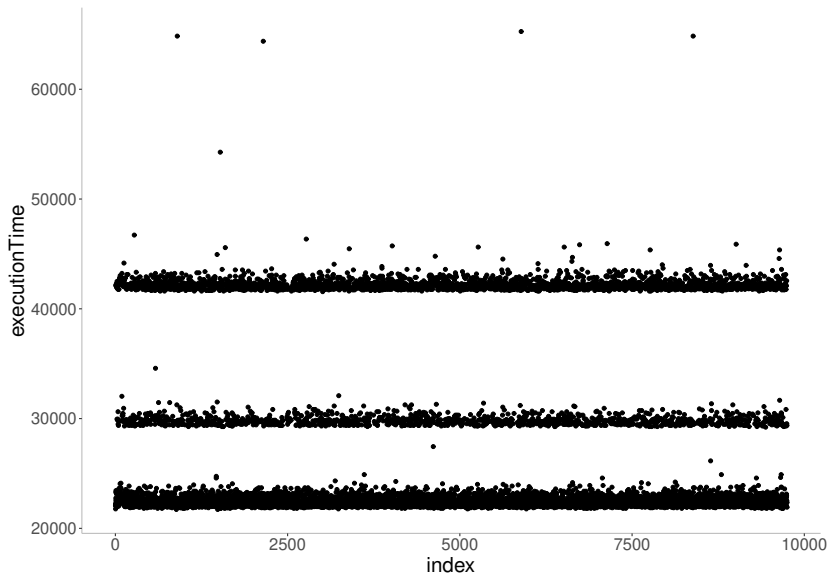


Fig. 8.1: The execution time sequence from the Markov Chain Test program used for estimating models. Times are given in nanoseconds.

Based on the means and standard deviations from Table 8.1, we can see that states 1 and 2 represent the program state with the lowest mean execution time, states 3 and 4 represent the middle program state and states 5 and 6 represent the highest program state and state 6 also some outliers. If we sum the values of columns 1-2, 3-4 and 5-6 for each row in Eq. (8.10), we see that for rows 1-5 they sum up to values similar to the corresponding transition probabilities in 8.9.

We see in Table 8.1 that the model is valid for all but one of the test sequences (test 8).

8.5.4 Video Decompression

A video is generated with images from the Tears of Steel open movie project[3]. The video is created with `ffmpeg` from frames 5000–8999 of the `1080bis-png` images[4]. The frame rate is set to 25 fps.

A trace is logged during decoding of the video with `ffmpeg` in native frame rate. The sequence of execution times from the decoding is displayed in Fig. 8.2. We consider the steady state, therefore the first 250 and the last 50 execution time measurements have been discarded, as outliers are seen on visual inspection. From the figures it is clear that the execution times are separated in distinct groups, the lower with execution times below 0.15 ms, accounting for approximately 57% of the samples, a slightly higher with execution times below 1 ms and a peak at around 0.45 ms, accounting for about 22% of the samples, a higher and more varying 10 ms accounting for approximately 19% of the samples, and the high with execution times above 22.5 ms accounting for less than 2% of the samples. These groups are considered as macrostates, and the transition probabilities between the states are displayed in Fig. 8.3. The execution time sequence has been separated outside of the framework and the transition probabilities in Fig. 8.3 are estimated directly from the sequence. The framework analysis is applied to each of these groups separately. The PFA_u values are with respect to the sequence used for model estimation.

[3] https://mango.blender.org/

[4] https://media.xiph.org/tearsofsteel/tearsofsteel-1080bis-png/

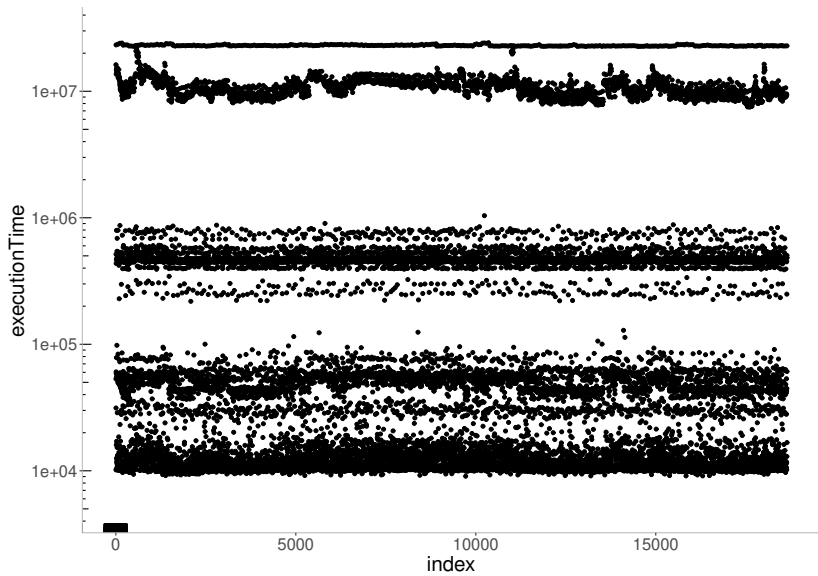


Fig. 8.2: The execution time sequence from the video decompression process. Times are given in naonseconds, log scale.

Macro state 1: Execution times below 0.15 ms

The execution times $c_i < 0.15$ ms are extracted from the video decompression log, resulting in a log of 10 538 samples. A Markov chain model is identified in the first two steps of the framework - the tree based cross validation approach described in Section 8.4.1, and fitting to the observations. The initial number of states is 20, and the resulting Markov model has 13 states. The data consistency criterion PFA_u for each state and for the entire model is calculated for the observations. The features and PFA_u values for the model is given in Table 8.2.

Macro state 2: Execution times in the range between 0.15 and 1 ms

The execution times $0.15 \leq c_i < 1$ ms consist of 4165 samples. The tree-based cross validation with 20 states in the initial Markov Model is applied and

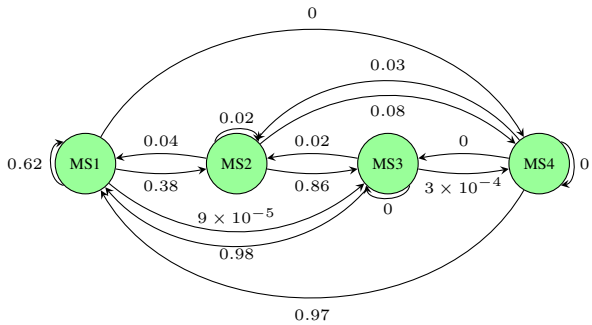


Fig. 8.3: Estimated transition probabilities between the macro states.

the resulting Markov Chain has 13 states. The state means and standard deviations of the estimated model, and the associated PFA_u values, are displayed in Table 8.3.

Macro state 3: Execution times in the range between 1 and 22.5 ms

The execution times $1 \leq c_i < 22.5$ ms are 3598 samples. The tree-based cross validation does not generally find a solution in this case - for many starting values the `depmixS4` fit function is unable to complete the expectation maximization step. Starting from 24 initial states, a solution with 14 states is found. The state means and standard deviations and corresponding PFA_u values for the model are listed in Table 8.4.

Macro state 4: Execution times above 22.5 ms

346 observations from the execution time sequence belong in macro state 4, $c_i \geq 22.5$ ms. The tree-based cross validation starting with 15 states identifies a model with 8 states. The state means and standard deviations and corresponding PFA_u values for the model estimated from the execution time sequence are shown in Table 8.5.

8.6 Discussion

The evaluation allows us to conclude that a Hidden Markov Model with Gaussian emission distributions can be appropriate to model execution time sequence data, and that the proposed framework can be used to identify and validate such a model.

The analysis of the Markov Chain test program shows that the methods can be used to estimate the number of modes, the transition matrix, means and standard deviations to fit the model. While the test program is constructed to display Markov Chain properties, we show that the execution time distributions in each state can be modeled by a combination of modes with Gaussian emission distributions. Compared to the video decompression test, the program has a simple structure and a small memory footprint.

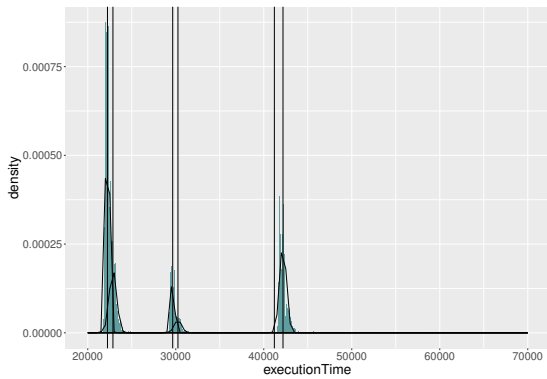
We also note that a Hidden Markov Model with Gaussian emission distributions appears to be valid in relation to the execution time sequences in the video decompression test.

The `depmixS4` methods used in the tree-based cross validation step are somewhat sensitive to the initial number of states, and the step may fail if this number is too large or too small. These methods can also fail if there are significant gaps between the execution time values, which is why the video decompression sequence is separated into macrostates. The framework could be expanded to manage separation into macrostates and find a suitable initial number of states.

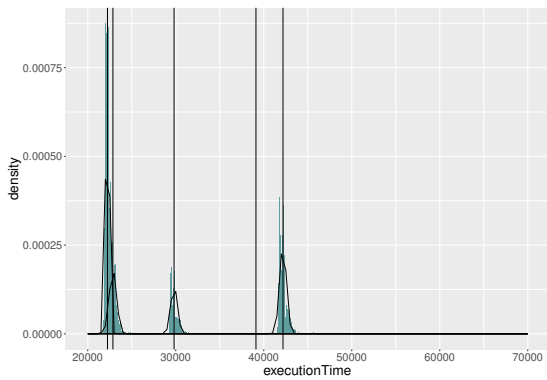
We also note that in some cases the number of states in the final models vary significantly. The heuristic algorithm for splitting the nodes could be adapted to evaluate a more exhaustive selection of possible splits, or replaced by an optimization algorithm such as for example simulated annealing [23].

Due to randomization utilized in many of the methods within the framework, different random seeds cause varying results. This is illustrated in Fig. 8.4, where the Gaussian distributions of two models are visualized on top of a normalized histogram of the sequence they are estimated from. The Gaussian distributions are scaled with their respective stationary distribution probabilities.

We have conducted preliminary tests with the validation step performed



(a)



(b)

Fig. 8.4: Normalized histograms of execution times (in nanoseconds) of the execution time sequence of the Markov chain test program. Two different estimated models from different random seeds are visualized with the Gaussian distributions of the states scaled with their respective stationary distribution probability, and their means displayed as vertical lines. In (a) we see the six state model from Section 8.5.3, and in (b) a five state model estimated with the framework applied to the same execution time sequence but initialized with another random seed.

with observations from running video decompression on another part of the "Tears of Steel" movie, that indicate that the identified model is not valid in this case. This may be due to input dependencies or cache related effects that cause the Gaussian distribution parameters and transition matrix to change over time and between runs.

Chen and Beltrame [10] show that effects of a random replacement cache can be described by an adaptive Markov Chain, and the ARM processor on the Raspberry Pi applies a pseudo-random cache replacement policy. Our methods derive a homogeneous Markov Chain, and if the model changes significantly during the sequence used for model parameter estimation, the model will not be valid, and this will be reflected in the resulting PFA_u values.

Finally, we note that cache-related jitter in our evaluations may be exaggerated by the `ftrace` process running simultaneously.

8.7 Conclusion and Future Work

This work proposed a measurement-based framework for probabilistic modeling of execution times of real-time applications. It presented an end-to-end workflow that first identifies the structure of a Markov Chain model and fits the probabilistic distributions to the collected execution time data, and finally validates the obtained model on the collected data based on a data consistency approach.

As with all measurement-based approaches, the application of this framework requires that the observations used at analysis are representative of the observations at runtime.

In order for the models to be useful in cases where full representativity of the observations at analysis time is not realistic to achieve, the methods described in this paper need to be complemented with (i) a method for providing a safe over-approximation of the execution time distribution, and (ii) a method for dynamically updating the model to reflect the effects on execution time patterns due to changes in input, program state or hardware state.

It is worth noticing that the proposed framework presents a consistent combination of different probabilistic tools, but it can include other techniques as alternatives. For example, the approach proposed in [20] for the identification

of the Markov model can be used in the first step of the proposed framework, as an alternative method. Further investigation on the tradeoffs among different techniques is needed, and it is deferred to future work.

The framework could be further extended and automated, e.g., by specifying required limits on the PFA_u values. If these are too close to 0 or 1, one can reject the model. Attempts can be made to identify new models in an iterative manner, until we find a model that is not rejected, or we reach an iteration limit and deem the proposed model not consistent with the observations.

Finally, this paper focused on the single use case of video decompression. Other use cases will be analyzed in the future to better understand and investigate benefits and drawbacks of different probabilistic tools that can be included in this framework.

Algorithm 8.2: Pseudo code describing the data consistency validation process.

Input: $\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*, \underline{c}\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*, \underline{c}$
Output: PFA_u

Function `DataConsistencyValidation`$(\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*, \underline{c})$**:**
  for $i \in 1 : M'$ **do**
    /* Generate M' trajectories of the same length as observations. */
    $traj1 \leftarrow$ `GenerateTraj`$(\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*, M', length(\underline{c}))$
    /* Calculate log likelihoods z_t and N $z_{t,j}$ for the samples as in Eqs. (8.5) and (8.6). */
    $simZ1[i] \leftarrow$ `CalcZ`$(traj1, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$
  /* Estimate $\mathbb{E}[z_t]$ and N $\mathbb{E}[z_{t,j}]$ across M' values for each round t. */
  $EZ \leftarrow$ `Mean`$(simZ1)$
  /* Estimate $\mathbb{E}[z_t]$ and N $\mathbb{E}[z_{t,j}]$ across M' values for each round t. */
  $VarZ \leftarrow$ `Var`$(simZ1)$
  for $i \in 1 : M$ **do**
    /* Generate M trajectories of the same length as observations. */
    $traj2 \leftarrow$ `GenerateTraj`$(\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*, M, length(\underline{c}))$
    /* Calculate log likelihoods z_t and N $z_{t,j}$ for the samples as in Eqs. (8.5) and (8.6). */
    $simZ2[i] \leftarrow$ `CalcZ`$(traj2, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$
    /* Calculate $M \times (N+1)$ Ts for z_t and $z_{t,j}$ as in Eq. (8.7) from simulated trajectories. */
    $Tsim[i] \leftarrow$ `CalcT`$(simZ2, EZ, VarZ)$
  /* Calculate log likelihoods z_t and N $z_{t,j}$ for the samples as in Eqs. (8.5) and (8.6). */
  $obsZ \leftarrow$ `CalcZ`$(\underline{c}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$
  /* Estimate the probability of S for the entire model and per state. */
  $PFA_u \leftarrow count(Tsim > Tobs)/M$
  return PFA_u

Table 8.1: State means and standard deviations (in nanoseconds) and corresponding PFA_u values for the model estimated from the training sequence.

State	**1**	**2**	**3**	**4**	**5**	**6**	**All**
mean	22240	22859	29652	30248	42185	41203	NA
stddev	222	420	221	409	383	9 190	NA
PFA_u							
test 1	0.22	0.21	0.18	0.18	0.94	0.44	0.31
test 2	0.52	0.49	0.33	0.33	0.14	0.56	0.87
test 3	0.38	0.37	0.24	0.24	0.43	0.05	0.02
test 4	0.28	0.27	0.19	0.18	0.53	0.20	0.19
test 5	0.36	0.36	0.22	0.22	0.42	0.22	0.24
test 6	0.14	0.14	0.12	0.14	0.75	0.10	0.06
test 7	0.26	0.26	0.18	0.18	0.67	0.38	0.19
test 8	0.00	0.00	0.01	0.01	0.58	0.00	0.00
test 9	0.58	0.58	0.43	0.39	0.01	0.44	0.89
test 10	0.55	0.53	0.42	0.40	0.02	0.62	0.92
test 11	0.27	0.26	0.19	0.20	0.76	0.24	0.17
test 12	0.43	0.43	0.29	0.26	0.04	0.31	0.41
test 13	0.48	0.47	0.25	0.23	0.03	0.31	0.50
test 14	0.74	0.73	0.50	0.46	0.01	0.69	0.99
test 15	0.28	0.28	0.19	0.20	0.58	0.31	0.14
test 16	0.29	0.28	0.21	0.21	0.41	0.14	0.56
test 17	0.37	0.37	0.21	0.21	0.09	0.13	0.76
test 18	0.36	0.36	0.24	0.23	0.37	0.12	0.41
test 19	0.28	0.28	0.21	0.21	0.53	0.59	0.61
test 20	0.19	0.20	0.18	0.19	0.81	0.16	0.11
train	0.40	0.39	0.28	0.26	0.45	0.48	0.94

Table 8.2: State means and standard deviations (in nanoseconds) and corresponding PFA_u values with the estimated model for macrostate 1.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**
mean	70 662	76 370	10 082	10 671	10 920	15 954	43 703
stddev	15 236	2 360	230	640	578	2 585	1 069
PFA_u	0.51	0.45	0.24	0.24	0.24	0.22	0.06

Table 8.2: State means and standard deviations (in nanoseconds) and corresponding PFA_u values with the estimated model for macrostate 1 (continued).

State	**8**	**9**	**10**	**11**	**12**	**13**	**All**
mean	54 343	28 470	12 927	31 295	62 083	39 794	NA
stddev	4 095	1 908	1 034	3 024	2 159	2 065	NA
PFA_u	0.22	0.17	0.24	0.23	0.47	0.10	0.02

Table 8.3: State means and standard deviations (in nanoseconds) and corresponding PFA_u values with the estimated model for macrostate 2.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**
mean	465 766	418 676	252 165	446 819	552 131	530 612	399 345
stddev	9 790	9 832	13 083	4 442	33 902	5 549	3 868
PFA_u	0.35	0.45	0.46	0.39	0.18	0.20	0.48

Table 8.3: State means and standard deviations (in nanoseconds) and corresponding PFA_u values with the estimated model for macrostate 2 (continued).

State	**8**	**9**	**10**	**11**	**12**	**13**	**All**
mean	773 326	681 703	481 032	301 732	586 839	452 460	NA
stddev	37 217	9 397	13 513	13 543	13 245	4 530	NA
PFA_u	0.24	0.20	0.31	0.46	0.17	0.37	0.37

Table 8.4: State means and standard deviations (in nanoseconds) and corresponding PFA_u values with the estimated model for macrostate 3.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**
mean	12 146 116	9 907 489	9 408 568	11 763 973	8 753 004	13 172 000	13 071 808
stddev	237 788	253 316	284 388	481 965	306 717	328 076	636 583
PFA_u	0.07	0.16	0.17	0.08	0.17	0.03	0.07

Table 8.4: State means and standard deviations (in nanoseconds) and corresponding PFA_u values with the estimated model for macrostate 3 (continued).

State	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**All**
mean	11 726 350	15 321 999	10 722 598	10 652 123	8 223 196	10 074 260	11 246 543	NA
stddev	276 281	2 759 114	288 845	456 994	332 875	286 002	314 261	NA
PFA_u	0.08	0.45	0.14	0.18	0.00	0.16	0.07	0.80

Table 8.5: State means and standard deviations (in nanoseconds) and corresponding PFA_u values with the estimated model for macrostate 4.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**All**
mean	22 979 652	22 881 488	23 863 163	22 786 818	22 733 574	23 365 667	23 094 934	23 198 326	NA
stddev	57 727	682 112	197 495	36 255	66 411	140 413	54 839	110 609	NA
PFA_u	0.39	0.36	0.37	0.36	0.35	0.32	0.30	0.38	0.57

Bibliography

[1] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 4–13, 1998.

[2] Luca Abeni and Giorgio Buttazzo. QoS guarantee using probabilistic deadlines. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 242–249, 1999.

[3] Luca Abeni and Giorgio Buttazzo. Stochastic analysis of a reservation based system. In *Int. Workshop on Parallel and Distributed Real-Time Systems*, volume 1, 2001.

[4] Luca Abeni, Daniele Fontanelli, Luigi Palopoli, and Bernardo Villalba Frías. A Markovian model for the computation time of real-time applications. In *IEEE international instrumentation and measurement technology conference (I2MTC)*, pages 1–6, 2017.

[5] Luca Abeni, Nicola Manica, and Luigi Palopoli. Efficient and robust probabilistic guarantees for real-time tasks. *Journal of Systems and Software*, 85(5):1147–1156, 2012.

[6] Sebastian Altmeyer, Liliana Cucu-Grosjean, and Robert I Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Real-Time Systems*, 51(1):77–123, 2015.

[7] Alan Burns and Robert I. Davis. A survey of research into Mixed Criticality Systems. *ACM Comput. Surv.*, 50(6), 2017.

[8] Alan Burns and Stewart Edgar. Predicting computation time for advanced processor architectures. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 89–96, 2000.

[9] Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Computing Surveys (CSUR)*, 52(1), 2019.

[10] Chao Chen and Giovanni Beltrame. An adaptive Markov model for the timing analysis of probabilistic caches. *ACM Trans. Design Automation of Electronic Systems (TODAES)*, 23(1):1–24, 2017.

[11] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *SIGCOMM Comput. Commun. Rev.*, 22(4):14–26, 1992.

[12] Tommaso Cucinotta, Luca Abeni, Mauro Marinoni, Alessio Balsini, and Carlo Vitucci. Reducing temporal interference in private clouds through real-time containers. In *IEEE International Conference on Edge Computing (EDGE)*, pages 124–131, 2019.

[13] Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J Cazorla. Measurement-Based Probabilistic Timing Analysis for multi-path programs. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 91–101, 2012.

[14] Robert Ian Davis and Liliana Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for Real-Time systems. *LITES: Leibniz Transactions on Embedded Systems*, pages 1–53, 2019.

[15] Robert Ian Davis and Liliana Cucu-Grosjean. A survey of probabilistic timing analysis techniques for Real-Time systems. *LITES: Leibniz Transactions on Embedded Systems*, 6(1):03–1–03:60, 2019.

[16] José Luis Díaz, Daniel F García, Kanghee Kim, Chang-Gun Lee, L Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella.

Stochastic analysis of periodic real-time systems. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 289–300, 2002.

[17] Jose Luis Diaz, Jose Maria Lopez, Manuel Garcia, Antonio M Campos, Kanghee Kim, and Lucia Lo Bello. Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 197–207, 2004.

[18] Bogdan Doytchinov, John Lehoczky, and Steven Shreve. Real-time queues in heavy traffic with Earliest-Deadline-First queue discipline. *Annals of Applied Probability*, pages 332–378, 2001.

[19] Stewart Edgar and Alan Burns. Statistical analysis of WCET for scheduling. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 215–224, 2001.

[20] Bernardo Villalba Frias, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. Probabilistic real-time guarantees: There is life beyond the i.i.d. assumption. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 175–186, 2017.

[21] David Griffin and Alan Burns. Realism in statistical analysis of Worst Case Execution Times. In *Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, 2010.

[22] Giordano A Kaczynski, Lucia Lo Bello, and Thomas Nolte. Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems. In *IEEE Int. Conf. on Emerging Technologies & Factory Automation (ETFA)*, pages 101–110, 2007.

[23] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[24] John P Lehoczky. Real-time queueing theory. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 186–195, 1996.

[25] Benjamin Lesage, David Griffin, Sebastian Altmeyer, Liliana Cucu-Grosjean, and Robert I Davis. On the analysis of random replacement

caches using static probabilistic timing methods for multi-path programs. *Real-Time Systems*, 54(2):307–388, 2018.

[26] Benjamin Lesage, David Griffin, Sebastian Altmeyer, and Robert I Davis. Static probabilistic timing analysis for multi-path programs. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 361–372, 2015.

[27] Andreas Lindholm, Dave Zachariah, Peter Stoica, and Thomas B Schön. Data consistency approach to model validation. *IEEE Access*, 7:59788–59796, 2019.

[28] Andreas Löfwenmark and Simin Nadjm-Tehrani. Fault and timing analysis in critical multi-core systems: A survey with an avionics perspective. *Journal of Systems Architecture*, 87:1–11, 2018.

[29] Yue Lu, Thomas Nolte, Iain Bate, and Liliana Cucu-Grosjean. A statistical response-time analysis of complex real-time embedded systems by using timing traces. In *IEEE Int. Symp. on Industrial and Embedded Systems (SIES)*, pages 43–46, 2011.

[30] Yue Lu, Thomas Nolte, Iain Bate, and Liliana Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 351–362, 2012.

[31] Yue Lu, Thomas Nolte, Johan Kraft, and Christer Norstrom. A statistical approach to response-time analysis of complex embedded real-time systems. In *IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 153–160, 2010.

[32] Yue Lu, Thomas Nolte, Johan Kraft, and Christer Norstrom. Statistical-based response-time analysis of systems with execution dependencies between tasks. In *IEEE Int. Conf. on Engineering of Complex Computer Systems*, pages 169–179, 2010.

[33] Nicola Manica, Luigi Palopoli, and Luca Abeni. Numerically efficient probabilistic guarantees for resource reservations. In *IEEE Int. Conf. on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8, 2012.

[34] Pau Martí, Josep M Fuertes, Gerhard Fohler, and Krithi Ramamritham. Improving Quality-of-Control using flexible timing constraints: metric and scheduling. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 91–100, 2002.

[35] Todd K. Moon. The Expectation-Maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[36] Luigi Palopoli, Daniele Fontanelli, Luca Abeni, and Bernardo Villalba Frias. An analytical solution for probabilistic guarantees of reservation based soft real-time systems. *IEEE Trans. Parallel and Distributed Systems*, 27(3):640–653, 2015.

[37] Eduardo Quinones, Emery D Berger, Guillem Bernat, and Francisco J Cazorla. Using randomized caches in probabilistic real-time systems. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 129–138, 2009.

[38] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. the IEEE*, 77(2):257–286, 1989.

[39] Takahiro Shinozaki. HMM state clustering based on efficient cross-validation. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, 2006.

[40] Václav Struhár, Moris Behnam, Mohammad Ashjaei, and Alessandro Vittorio Papadopoulos. Real-time containers: A survey. In *Workshop on Fog Computing and the IoT (Fog-IoT)*, volume 80, pages 7:1–7:9, 2020.

[41] Aida Čaušević, Alessandro Vittorio Papadopoulos, and Marjan Sirjani. Towards a framework for safe and secure adaptive collaborative systems. In *IEEE Annual Computer Software and Applications Conf. (COMPSAC)*, volume 2, pages 165–170, 2019.

[42] Ingmar Visser, Maarten Speekenbrink, et al. depmixs4: an R package for hidden Markov models. *Journal of Statistical Software*, 36(7):1–21, 2010.

[43] Reinhard Wilhelm. Mixed Feelings About Mixed Criticality (Invited Paper). In *Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 63, pages 1:1–1:9, 2018.

Chapter 9

Paper B
Adaptive Runtime Estimate of Task Execution Times Using Bayesian Modeling.

Anna Friebe, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte. In IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2021.

Abstract

In the recent works that analyzed execution-time variation of real-time tasks, it was shown that such variation may conform to regular behavior. This regularity may arise from multiple sources, e.g., due to periodic changes in hardware or program state, program structure, inter-task dependence or inter-task interference. Such complexity can be better captured by a Markov Model, compared to the common approach of assuming independent and identically distributed random variables. However, despite the regularity that may be described with a Markov model, over time, the execution times may change, due to irregular changes in input, hardware state, or program state. In this paper, we propose a Bayesian approach to adapt the emission distributions of the Markov Model at runtime, in order to account for such irregular variation. A preprocessing step determines the number of states and the transition matrix of the Markov Model from a portion of the execution time sequence. In the preprocessing step, segments of the execution time trace with similar properties are identified and combined into clusters. At runtime, the proposed method switches between these clusters based on a Generalized Likelihood Ratio (GLR). Using a Bayesian approach, clusters are updated and emission distributions estimated. New clusters can be identified and clusters can be merged at runtime. The time complexity of the online step is O(N^2 + NC) where N is the number of states in the Hidden Markov Model (HMM) that is fixed after the preprocessing step, and C is the number of clusters.

9.1 Introduction

The characterization of the execution time of a real-time task is an important step towards analyzing the schedulability of a real-time system. The execution-time characterization usually focuses on the Worst-Case Execution Time (WCET), allowing for the analysis of hard real-time guarantees (for more details see [33, 17]). On the other hand, hardware acceleration features, multi-core systems [20] and complex, interacting tasks, e.g., in mixed criticality systems [3, 32], pose several challenges in achieving tight bounds on the WCET and Worst-Case Response-Time (WCRT) [17].

In the recent decades, probabilistic approaches have been proposed in relation to execution time estimates. The main purpose of the probabilistic approaches is to derive a more realistic distribution of the execution-time values, lowering upon the over-provisioning when one considers the worst-case values, while still considering Quality of Service (QoS) [6] or Quality of Control (QoC) [26]. The majority of this work considers estimating the probabilistic WCET (pWCET) distribution, that upper-bounds the execution time distributions of all valid scenarios and feasible sequences of repeated program execution [9, 10, 5]. Measurement-based techniques based on Extreme Value Theory (EVT) [4, 12, 15] require that extreme values of the execution time distribution are independent and that the measurements contain samples from the worst case distribution [30, 18]. As an upper bound, the pWCET may still be very pessimistic compared to the average execution time, and compared to the upper bound of the execution time distributions of scenarios that are valid in a more limited context of task execution that involves hardware and software state as well as input.

In some cases the entire distribution may be relevant, and not only the upper bound based on the distribution's tail. One such case is where QoS/ QoC adaptation can be utilized. Tasks can allow for different QoS/ QoC levels as proposed by Lu et al.[21]. In a robotic application, the robot's speed can be adjusted to allow for a lower frequency control loop. In these cases, the deadline miss probability can be kept sufficiently low with task adaptation. This can also relax the requirement to capture samples from the most extreme conditions in the analysis stage, provided the adaptation options are satisfactory.

In this paper, we address the problem of runtime estimation of execution-time distribution, analyzing the execution trace of a task **at runtime**. More specifically, in a preprocessing step, the number of states and the transition matrix of the Hidden Markov Model (HMM) are derived from a portion of the execution time sequence. Segments within this sequence that are similar are identified. Here we use the Generalized Likelihood Ratio (GLR), a measure for the likelihood that the segments are generated from the same HMM. Similar segments are combined into clusters, to form HMMs with differing emission distributions. At runtime, the algorithm switches between these HMMs depending on similarity with the current segment of the execution time trace. New HMMs can be created and the emission distributions updated. The complexity of the proposed runtime adaptive algorithm for the estimation of task distributions is $\mathcal{O}(N^2 + NC)$ where N is the number of states in the HMM that is fixed after the preprocessing step, and C is the number of clusters. The proposed approach has the potential for being used for the assessment of several real-time system properties, but such an investigation is beyond the scope of this work.

The remainder of this paper is organized as follows. Related work is outlined in Section 9.2. In Section 9.3, we describe the system-model assumptions, along with definitions and mathematical background used in the paper. Then, in Section 9.4 we describe the derivation of the initial HMM in the preprocessing step, which is followed by Section 9.5, the description of the method for online model-parameter adaptation. The evaluation is described in Section 9.6, and the paper is concluded in Section 9.7.

9.2 Related Work

Two major surveys on the Probabilistic Timing Analysis [9] and the Probabilistic Schedulability Analysis [8] of real-time systems have been conducted by Davis and Cucu-Grosjean, while a taxonomy and survey on pWCET analysis and associated methods was provided by Cazorla et al. [5]. We further describe the state-of the art in measurement-based methods, where the contributions of this paper fall into.

Measurement-Based Probabilistic Timing Analysis was introduced by

Cucu-Grosjean [7], based on previous work related to the use of Extreme Value Theory (EVT) [4, 12, 15]. EVT is applied to find the pWCET, an upper bound on the probability of exceeding each possible execution time value. Methods based on EVT require that extreme values of the execution time distribution are independent [30, 18]. EVT-based techniques have also been applied in order to estimate upper bounds on response time distributions [25, 24, 22, 23].

Moving from extreme values, and focusing on estimates of the full execution time distribution, the distribution of a visual task in a robotic application has been modeled as a HMM with discrete emission distributions by Frías et al. [1, 13]. HMMs can capture the regularity and dependability in the task execution, that may arise from different sources, e.g., sensed input, periodic nature of task interactions, or the algorithms being used in the tasks.

Friebe et al. [14] proposed an approach to estimate the execution time distribution using HMMs with Gaussian emission distributions, and proposed an automatic way of estimating the number of states in HMM from the execution trace. The methods from [14] are utilized for HMM fitting in the preprocessing step.

In all these approaches, the structure of the HMM and the emission distributions are learned in an offline phase, based on existing logged data. However, although in the base case the execution times may be characterized with a Markov Model, throughout the task's life cycle the model accuracy may deteriorate due to different irregularities such as changes in input, hardware, or program state. If the observations used for fitting the HMM are not fully representative of the runtime observations, the model may also be inaccurate. In Frías et al. [13], two separate experiments are performed, for a clean and a noisy track respectively, and these give rise to two different Markov Models, with notably different bandwidth requirements. In order to apply these methods for tasks where the context affecting the execution time distribution may change, an adaptive approach is necessary. In this work, we assume that a preprocessing phase is conducted where the HMM fitting is performed as in previous work [14], but we propose a runtime Bayesian adaptation method to continuously refine the execution time model based on the new observations.

Lu et al. [21] propose a Feedback Control Real-Time Scheduling (FCS) architecture, including a Monitor, a Controller and a QoS Actuator. An adaptive

estimate of the execution time distribution could allow for the Monitor to predict the deadline miss probability rather than measuring the past deadline miss ratio. To the best of our knowledge, no adaptive runtime estimates of execution time distributions have previously been proposed.

In the proposed method, segments of the execution time trace are considered, and the similarity measure is defined considering the HMM as a whole. An alternative could have been to consider each execution time sample separately, and adapt and add states to a single HMM while updating the transition matrix. This could be be achieved by considering novelty detection such as in Gruhl et al. [16] in combination with a HMM update mechanism. We hypothesize that the context affecting a task's execution time distribution can change suddenly, and that the task can be affected in a similar way in several segments during the execution. Therefore we have chosen to consider execution time segments, and to enable switching betweeen clusters. In the proposed Bayesian model, the emission distributions are Gaussian with unknown mean and precision. An alternative could have been to model the emission distributions as Gaussian with unknown mean but fix the precision estimates in the preprocessing step. Due to the risk of underestimating the variance and thus the tail width, this option was not chosen.

9.3 System Model and Definitions

In this section, a task model with irregular execution-time variability is outlined. A Bayesian model for estimating the execution-time distribution from observations is described. A measure of similarity, Generalized Likelihood Ratio (GLR), for the Bayesian models is presented. This measure is used in the preprocessing and adaptive steps to determine points where the execution-time distribution changes, and to find similar segments of the execution-time trace.

Notation. We denote sequences with parentheses, $()$, and sets with braces, $\{\}$. The estimate of a quantity x is indicated in the following as $\hat{x}$. Table 9.1 lists the main symbols used in the paper.

Notation	Description
$cs = (c_1, c_2, \ldots, c_t)$	Execution-time sequence
N	Number of states in the Markov Model
$\mathcal{M} = \{m_1, m_2, \ldots, m_N\}$	Set of Markov states
$\mathcal{P}$	$N \times N$ state transition matrix
$\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$	Set of execution-time distributions
s_j	Contiguous segment of cs
$\{\mu_{nj}, \sigma^2_{nj}\}$	Mean and variance of state m_n in segment s_j
$S_k = \{s_j, \ldots\}$	Cluster, set of segments
γ_{ni}	Occupancy probability of state m_n in segment s_i
$\hat{\mathbf{a}}[0]_{jn}$	Estimated number of observations in state m_n and segment s_j
$\hat{\mathbf{a}}[1]_{jn}$	Estimated sum of observations in state m_n and segment s_j
$\hat{\mathbf{a}}[2]_{jn}$	Estimated sum of squared observations in state m_n and segment s_j
$\hat{\mu}_{nk}$	Estimated mean of state n and cluster S_k
$\hat{\nu}_{nk}$	Estimated variance of state n and cluster S_k
$NG(\mu, \lambda)$	Normal-Gamma distribution
$\mu_0, \kappa_0, \alpha_0, \beta 0$	Prior hyperparameters of NG
$\mu_L, \kappa_L, \alpha_L, \beta_L$	Posterior hyperparameters of NG
$D = (x_1, \ldots, x_L)$	Observation sequence of length L
$\mathrm{GLR}(S_k, S_l)$	Generalized Likelihood Ratio between clusters S_k and S_l
ℓ_k	Log-likelihood of cluster S_k
$nPseudoObs$	Number of pseudo observations in prior construction
$\mathcal{GP}(m(x), k(x, x'))$	Gaussian process with mean m and kernel k

Table 9.1: Important notation used in this work.

9.3.1 Task Model

We consider a periodic task, that generates a sequence of jobs. The sequence of execution times of the jobs is $cs = (c_1, c_2, \ldots, c_t)$. We assume that the execution-time sequence cs can be characterized by a Markov model, described by the set $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$, where

- $\mathcal{M} = \{m_1, m_2, \ldots, m_N\}$ is the set of N states, with m_n, $n \in \mathbb{N}$.
- $\mathcal{P}$ is the $N \times N$ state transition matrix, where the element $p_{a,b}$ represents the conditional probability $\mathbb{P}(X_{i+1} = m_b | X_i = m_a)$ of being in state m_b at round $i + 1$, given that at round i the state is m_a.
- $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$ is the set of execution-time distributions, or

emission distributions related to respective state. In this paper, these are modeled as Gaussian distributions with mean μ_n, and variance σ_n^2, i.e., $C_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$.

In the sequence cs, we assume that N and $\mathcal{P}$ remain unchanged, but at a finite number of points in the sequence, referred to as points of cluster change, the parameters $\{\mu_n, \sigma_n^2\}$ may take new (different) values. For this purpose we introduce another index, j, to explicitly indicate the dependency on time. The parts of the sequence where $\{\mu_n, \sigma_n^2\}$ remain constant are referred to as *segments* s_j. In each segment s_j, the mean and variance are denoted $\{\mu_{nj}, \sigma_{nj}^2\}$. A set $S_k = \{s_j, \ldots\}$ of non-adjacent segments with the same values of $\{\mu_{nj}, \sigma_{nj}^2\}$ are referred to as a *cluster*. An illustration is shown in Fig. 9.1.

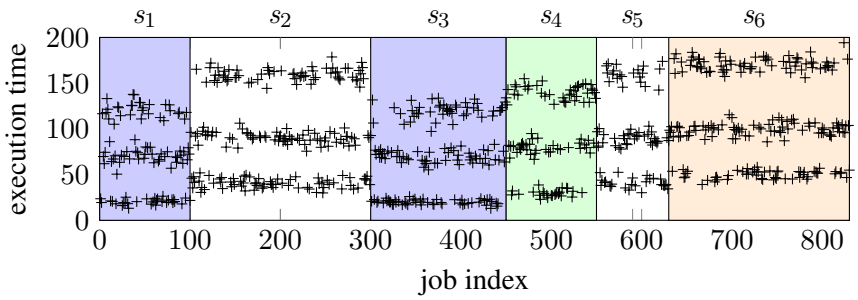


Fig. 9.1: An execution time sequence separated into six *segments* $(s_1, s_2, s_3, s_4, s_5, s_6)$ and four *clusters* (S_1, S_2, S_3, S_4), where $S_1 = \{s_1, s_3\}$, $S_2 = \{s_2, s_5\}$, $S_3 = \{s_4\}$, and $S_4 = \{s_6\}$.

9.3.2 Estimating Sufficient Statistics

The Markov Model $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$ can be estimated by the use of the Forward-Backward algorithm [29] in combination with the Expectation Maximization algorithm [27]. The number of states N can be determined as described in Section 9.4. Given this information and an execution time sequence or segment, the state occupancy probabilities γ_{ni} can be obtained for each state m_n and execution time observation cs_i using the Forward-Backward algorithm.

The occupancy probabilities are used to calculate sufficient statistics (presented in [14, 31]) for each segment s_j and state n. Sufficient statistics are a compact way of storing the information needed to estimate the Gaussian emission distribution of each state within the segment, and are also used when updating the Bayesian model. The sufficient statistics for a Gaussian distribution are: (i) $\hat{\mathbf{a}}[0]$, an estimate of the number of observations in the state, (ii) $\hat{\mathbf{a}}[1]$, an estimate of the sum of the observations in the state, and (iii) $\hat{\mathbf{a}}[2]$, an estimate of the sum of the squared observations in the state.

$$\hat{\mathbf{a}}[0]_{jn} = \sum_{i, c_i \in s_j} \gamma_{ni}, \tag{9.1}$$

$$\hat{\mathbf{a}}[1]_{jn} = \sum_{i, c_i \in s_j} c_i \gamma_{ni}, \tag{9.2}$$

$$\hat{\mathbf{a}}[2]_{jn} = \sum_{i, c_i \in s_j} c_i^2 \gamma_{ni}. \tag{9.3}$$

9.3.3 Bayesian Model

In a Bayesian approach, a conjugate distribution is a distribution where the posterior probability $p(\Theta|D)$ of the parameter Θ given observations D, takes the same functional form as the prior distribution $p(\Theta)$ [2]. For a Gaussian probability distribution with unknown mean μ and precision $\lambda = 1/\sigma^2$, the conjugate distribution is a Normal-Gamma distribution [28], denoted as $NG(\mu, \lambda)$. When we have a prior distribution of μ and λ as given by

$$\begin{aligned} p(\mu, \lambda) = NG(\mu, \lambda | \mu_0, \kappa_0, \alpha_0, \beta_0) \triangleq \\ \mathcal{N}(\mu | \mu_0, (\kappa_0 \lambda)^{-1}) Ga(\lambda | \alpha_0, rate = \beta_0), \end{aligned} \tag{9.4}$$

and observations $D = (x_1, \ldots, x_L)$, the posterior probability distribution can be computed as:

$$p(\mu, \lambda | D) = NG(\mu, \lambda | \mu_L, \kappa_L, \alpha_L, \beta_L), \tag{9.5}$$

$$\mu_L = \frac{\kappa_0 \mu_0 + \sum_{i=1}^{L} x_i}{\kappa_0 + L}, \tag{9.6}$$

$$\kappa_L = \kappa_0 + L, \tag{9.7}$$

$$\alpha_L = \alpha_0 + L/2, \tag{9.8}$$

$$\beta_L = \beta_0 + \frac{1}{2}\left(\sum_{i=1}^{L}(x_i - \bar{x})^2 + \frac{\kappa_0 L(\bar{x} - \mu_0)^2}{(\kappa_0 + L)}\right). \tag{9.9}$$

Given that we are not certain of which state each observation c_i belongs to, we rewrite Eqs. (9.6) to (9.9), using the sufficient statistics in Eqs. (9.1) to (9.3), yielding:

$$\mu_L = \frac{\kappa_0\mu_0 + \hat{\mathbf{a}}[1]}{\kappa_0 + \hat{\mathbf{a}}[0]}, \tag{9.10}$$

$$\kappa_L = \kappa_0 + \hat{\mathbf{a}}[0], \tag{9.11}$$

$$\alpha_L = \alpha_0 + \hat{\mathbf{a}}[0]/2, \tag{9.12}$$

$$\beta_L = \beta_0 + \frac{1}{2}\left(\hat{\mathbf{a}}[2] - \frac{\hat{\mathbf{a}}[1]^2}{\hat{\mathbf{a}}[0]} + \frac{\kappa_0\hat{\mathbf{a}}[0](\frac{\hat{\mathbf{a}}[1]}{\hat{\mathbf{a}}[0]} - \mu_0)^2}{(\kappa_0 + \hat{\mathbf{a}}[0])}\right), \tag{9.13}$$

where we excluded the indices j, n for readability, and we used $\hat{\mathbf{a}}[0]_{jn}$ to estimate L, $\hat{\mathbf{a}}[1]_{jn}$ to estimate $L\bar{x}$ and $\hat{\mathbf{a}}[2]_{jn}$ to estimate $L\bar{x}^2$. Eqs. (9.10) to (9.13) describe the Normal Gamma parameters for a state and segment.

The initial hyperparameters can be conceptualized as derived from a number of pseudo-observations, with the mean μ_0 derived from κ_0 observations and the precision λ_0 derived from $2\alpha_0$ observations with mean μ_0 and sum of squared deviations $2\beta_0$.

Since the Normal-Gamma distribution is the conjugate distribution, segments can be added and the posterior hyperparameters updated incrementally by substituting the existing posterior hyperparameters with the prior hyperparameters. In this way the posterior Normal-Gamma distribution for a cluster is constructed by incrementally updating the hyperparameters for each segment in the cluster. Similarly, segments can be removed from the estimate by updating

the hyperparameters according to:

$$\mu_L = \frac{\kappa_0\mu_0 - \hat{\mathbf{a}}[1]}{\kappa_0 - \hat{\mathbf{a}}[0]}, \tag{9.14}$$

$$\kappa_L = \kappa_0 - \hat{\mathbf{a}}[0], \tag{9.15}$$

$$\alpha_L = \alpha_0 - \hat{\mathbf{a}}[0]/2, \tag{9.16}$$

$$\beta_L = \beta_0 - \frac{1}{2}\left(\hat{\mathbf{a}}[2] - \frac{\hat{\mathbf{a}}[1]^2}{\hat{\mathbf{a}}[0]} + \frac{\kappa_0\hat{\mathbf{a}}[0](\frac{\hat{\mathbf{a}}[1]}{\hat{\mathbf{a}}[0]} - \mu_0)^2}{(\kappa_0 - \hat{\mathbf{a}}[0])}\right). \tag{9.17}$$

Note that the posterior predictive distribution for a single point prediction is the Student's t-distribution as described by Murphy [28]:

$$p(x|D) = t_{2\alpha_L}\left(\frac{\beta_L(\kappa_l + 1)}{\alpha_L\kappa_L}\right). \tag{9.18}$$

9.3.4 GLR Between Sets of Segments

The GLR of two sets of observations can be used to determine whether the sets are likely to belong to a joint distribution or if it is more likely that they belong to distinct distributions. This is done by calculating the ratio of the probability of the observations under the joint model and the product of the probabilities of the observations under distinct models. The GLR measure is central to the proposed approach and used in the preprocessing as well as the adaptive step.

The GLR between two execution time segment sets S_k and S_l, and the union of the sets given as $S_{k\cup l}$, using log-likelihoods (indicated with ℓ), is [19]:

$$\mathrm{GLR}(S_k, S_l) = \ell_{k\cup l} - (\ell_k + \ell_l). \tag{9.19}$$

The posterior predictive distribution for m new observations given the Normal-Gamma prior is given in Murphy [28] as

$$p(D_{new}|D) = \frac{\Gamma(\alpha_{n+m})}{\Gamma(\alpha_n)}\frac{\beta_n^{\alpha_n}}{\beta_{n+m}^{\alpha_{n+m}}}\sqrt{\frac{\kappa_n}{\kappa_{n+m}}}(2\pi)^{m/2}. \tag{9.20}$$

were Γ represents the gamma function.

We use the same prior distribution for the two segment sets and for the union of the sets. Evaluating the posterior probability of the observations in a state of a segment set S_k under the posterior predictive distribution calculated from the same observations gives the log-likelihood using estimates from Eqs. (9.11) to (9.13) as:

$$\begin{aligned}\ell_k = \; & \log\Gamma(\alpha_{2k}) - \log\Gamma(\alpha_k) + \alpha_k \log\beta_k - \\ & \alpha_{2k}\log\beta_{2k} + \frac{1}{2}(\kappa_k - \kappa_{2k}) + \frac{\hat{\mathbf{a}}[0]_k}{2} 2\pi.\end{aligned} \tag{9.21}$$

The last terms cancel out in the GLR, and the resulting equation for two segment sets and a state is:

$$\begin{aligned}\mathrm{GLR}(S_k, S_l) = & \log\Gamma(\alpha_{2(k\cup l)}) - \log\Gamma(\alpha_{(k\cup l)}) + \alpha_{k\cup l}\log\beta_{k\cup l} \\ & - \alpha_{2(k\cup l)}\log\beta_{2(k\cup l)} + \frac{1}{2}(\kappa_{k\cup l} - \kappa_{2(k\cup l)}) \\ & - \log\Gamma(\alpha_{2k}) + \log\Gamma(\alpha_k) - \alpha_k\log\beta_k \\ & + \alpha_{2k}\log\beta_{2k} - \frac{1}{2}(\kappa_k - \kappa_{2k}) \\ & - \log\Gamma(\alpha_{2l}) + \log\Gamma(\alpha_l) - \alpha_l\log\beta_l \\ & + \alpha_{2l}\log\beta_{2l} - \frac{1}{2}(\kappa_l - \kappa_{2l})\end{aligned} \tag{9.22}$$

The GLR of two segment sets is estimated as the sum of the GLRs over the states. We use GLRs based on log-likelihoods, so this is equivalent to multiplication of the GLR measure as described by Liu and Kubala [19].

9.4 Preprocessing Step

The preprocessing step is performed on an execution time sequence where we expect to capture the regular variation of execution times. The preprocessing step identifies:

1. The number of states N and transition matrix $\mathcal{P}$ of the cluster HMMs.
2. The segments and clusters within this execution time sequence.

3. The sufficient statistics for the HMM states of each cluster.

A HMM is fitted to the execution time sequence, using the tree-based cross validation approach described in [14]. This fitting process provides the number of states and transision matrix.

The normal distribution parameters μ, σ in the HMM from the preprocessing step are used in combination with a number of pseudo observations, $nPseudoObs$, to create initial prior Normal-Gamma distributions as:

$$\mu_0 = \mu, \tag{9.23}$$
$$\kappa_0 = nPseudoObs, \tag{9.24}$$
$$\alpha_0 = \frac{nPseudoObs}{2}, \tag{9.25}$$
$$\beta_0 = \alpha_0 \cdot \sigma^2. \tag{9.26}$$

The number $nPseudoObs$ is chosen for each state in relation to the stationary probability of the state.

9.4.1 Finding Points of Cluster Change

For a sequence of execution time observations, we want to find the set of points where the model parameters change.

Finding One Point of Model Change

Initially, we consider the simpler problem of finding one point of model change in a sequence. Given a starting index x_{start} and a stopping index x_{stop} within the sequence, we aim to find the index x_{split} that minimizes the $GLR(x)$, defined as

$$\mathrm{GLR}(x) = \mathrm{GLR}(\{s_{x-}\}, \{s_{x+}\}) \tag{9.27}$$
$$s_{x-} = (c_{x_{start}}, c_{x_{start}+1}, \ldots, c_x) \tag{9.28}$$
$$s_{x+} = (c_{x+1}, c_{x_{split}+1}, \ldots, c_{x_{stop}}) \tag{9.29}$$
$$x_{split} = \arg\min_x \mathrm{GLR}(x) \tag{9.30}$$

where the segments s_{x-} and s_{x+} indicate the segments before and after x. The optimization is performed Bayesian Optimization as implemented in the Python library GpyOpt[1], where BayesianOptimization is configured with a Radial Basis Function kernel and Expected Improvement as acquisition function. Posterior predictive Student's t-distributions are derived for s_{x-}, s_{x-} and for their union. The log-likelihoods for these segments are calculated by applying the Forward-Backward algorithm. The transition matrix $\mathcal{P}$ in taken from the fitted HMM, but the posterior predictive distributions are used as emission distributions.

If the resulting $\mathrm{GLR}(x_{split})$ is lower than a given GLR_{limit}, x_{split} is considered to be a point of model change.

Finding Several Points of Model Change

In order to find several points of model change within an execution time sequence cs, we apply the method described in Section 9.4.1 for the entire sequence, $x_{start} = 1$, $x_{stop} = t$. Recursively, the method is applied for the sequences with $x_{start} = 1$, $x_{stop} = x_{split}$ and $x_{start} = x_{split} + 1$, $x_{stop} = t$, and further, similarly to a binary search approach, until one of the following stopping criteria are met:

1. The resulting $\mathrm{GLR}(x_{split})$ is above the given GLR_{limit};

2. The length of the subsequence is below a minimum length between splitting points.

9.4.2 Segment Clustering

The segments are clustered into sets using an approach similar to the Leader-Follower Clustering described by Duda et al. [11]. The longest segment is added as the first cluster. For each segment, in order of decreasing segment length, the cluster that gives the maximum GLR is found as outlined in Section 9.3.4. If the GLR between the segment and the closest cluster is large

[1] https://sheffieldml.github.io/GPyOpt/

enough, the segment is merged into the cluster, otherwise a new cluster is created. The threshold has been set to 10 times the threshold used in finding the points of model change.

9.5 Online Model Adaptation

In the runtime process, a sliding window is considered. A simplified flowchart of the algorithm is available in Fig. 9.2.

The sliding window has a length of $T = a \cdot step$. Here, a and $step$ are integers, and $step$ is the size of the sliding window movement at each step. We assume that a starting cluster at the beginning of the window is known. The sliding window hyperparameters are estimated by calculating the posterior distribution using Eqs. (9.10) to (9.13) with the initial prior distribution. Sufficient statistics $\hat{\mathbf{a}}[0]$, $\hat{\mathbf{a}}[1]$ and $\hat{\mathbf{a}}[2]$ are derived using the Forward-Backward algorithm [29]. The emission distributions for the states are generalized Student's t-distributions, the posterior predictive distributions of the cluster at the start of the window as given in Eq. (9.18). The prior distribution is chosen as outlined in Section 9.4, Eqs. (9.23) to (9.26).

A number of GLR thresholds are used in the process, to determine whether a cluster change shall be made, if a new cluster shall be created, or if the current cluster shall be merged with another. In addition we use different thresholds for preprocessing clusters compared to newly created clusters, where we require a closer match with newly created clusters. One reason for this is that new clusters can be dominated by the prior distribution, and for this reason are more likely to have a high GLR when compared to each other. We base all thresholds on the threshold used for finding points of model change in the preprocessing step. Different multiplicative factors are applied, according to Table 9.2.

9.5.1 Determining if there is a Cluster Change in the Window

The GLR is estimated of the sliding window and the starting cluster distributions. If this is below a threshold $slidingLimit$, we move into the right column of Fig. 9.2. A segment $clusterFindSegment$ of length T around the endpoint of the sliding window is considered. A Normal-Gamma distribution for this

Threshold	Purpose	Factor
slidingLimit	Check if cluster change in slidingwindow	1
newClusterLimit	Create new cluster	2
changePreLimit	Change to a preprocessing cluster	1
mergeClusterPreLimit	Merge current with preprocessing cluster	1.5
mergeClusterLimit	Merge current with closest cluster	1

Table 9.2: Thresholds used in the adaptive process and the multiplicative factor to the threshold used in finding points of model change in the preprocessing step.

segment is calculated using Eqs. (9.10) to (9.13) and the initial prior distribution. Sufficient statistics are derived with the Forward-Backward algorithm using the generalized Student's t-distribution as the posterior predictive of the initial prior distribution. The point of cluster change and the cluster at the end of the sliding window are determined as outlined in Algorithm 9.1.

Determining the Point of Cluster Change: The sliding window is divided into chunks that are considered from the endpoints of the sliding window. The GLR of the starting cluster with the first chunk is compared to the GLR of $closestClusterAll$ with the last chunk. Iteratively, the chunk with the highest GLR is added to the start or end sections of the sliding window, and the next chunk from the appropriate side is considered, until all chunks are added to either side.

9.5.2 Updating the Sliding Window and Clusters

If a cluster change has taken place, we continue downwards in the right column of Fig. 9.2. A new sliding window is created from the endpoint of the current, using posterior student distributions of the new cluster to calculate the sufficient statistics. Posterior distributions and sufficient statistics for the previous and new clusters are updated with the sliding window segments prior and after the cluster changing point.

If there is no need for cluster change, we proceed in the left column of Fig. 9.2. The sliding window is advanced with the step size. The distributions are updated by removing the sufficient statistics for the step no longer in

Algorithm 9.1: Pseudocode describing the process of finding the potential point of change and the new cluster.

Input: Current cluster at the beginning of the sliding window, preprocessing and adaptive clusters, sliding window and cluster finding segment with Normal-Gamma distributions.

Output: Point of cluster change and current cluster at end of sliding window.

Function `ClusterChange`(clusters, preClusters, clusterFindSegment, slidingWindow, currentCluster) **:**

- closestClusterAll $\leftarrow \arg\max_{c \in \text{clusters}} GLR(c, \text{clusterFindSegment})$
- potentialChangePoint $\leftarrow$ FINDCHANGEPOINT(slidingWindow, currentCluster, closestClusterAll)
- testEndSegment $\leftarrow$ slidingWindow[potentialChangePoint:end]
- testNGAll $\leftarrow$ POSTERIORNG(closestClusterAll, testEndSegmentNG)
- testGLRAll $\leftarrow GLR$(closestClusterAll, testNGAll)
- **if** testGLRAll $<$ newClusterLimit **then**
  - newCluster $\leftarrow$ CREATECLUSTER(testEndSegment)
  - **return** potentialChangePoint, newCluster
- closestClusterPre $\leftarrow \arg\max_{c \in \text{preClusters}} GLR(c, \text{clusterFindSegment})$
- testNGPre $\leftarrow$ POSTERIORNG(closestClusterPre, testEndSegmentNG)
- testGLRPre $\leftarrow GLR$(closestClusterPre, testNGPre)
- **if** testGLRPre $>$ changePreLimit **then**
  - **return** potentialChangePoint, closestClusterPre
- **return** potentialChangePoint, closestClusterAll

the sliding window, and adding those for the new step, according to Eqs. (9.10) to (9.13) and Eqs. (9.14) to (9.17). The updated cluster is compared with the other existing clusters. If the GLR of the current cluster and the closest cluster is large enough the clusters are merged.

9.5.3 Complexity Analysis

The computation of sufficient statistics with the Forward-Backward method has a time complexity of $\mathcal{O}(N^2 L)$, where N is the number of states and L is the length of the considered section. For each window, L is bounded by $2T$, as we may need to calculate sufficient statistics for the $clusterFindSegment$ when

a cluster change is considered and for a new sliding window in the event of cluster change.

The GLR calculations are summed over the states, and we find the maximum GLR among all clusters, resulting in a total time complexity of $\mathcal{O}(NC)$, where N is the number of states and C is the number of clusters.

The total time complexity of the adaptive step is $\mathcal{O}(N^2 + NC)$, where N is the number of states in the HMM, fixed after the preprocessing step, and C is the number of clusters.

9.6 Evaluation

9.6.1 Goal of the Evaluation

In the following experiments[2], we first generated the execution samples according to the predefined ground truth model, and then we performed the proposed method on the execution samples in order to estimate the posterior distribution. The goal of the experiments was to investigate the accuracy of the estimated posterior distribution having the ground truth model as the reference. By using synthetic data in the evaluations we can make comparisons to the ground truth distributions. Comparisons are made by calculating the Kullback-Leibler (KL) divergence, as will be further outlined below.

In the experiments, we distinguish the two main steps, the preprocessing step of the method – where the initial execution sample is analyzed in an offline manner – and the adaptive process—where the estimated parameters, from the preprocessing step, are adaptively modified online in order to account for the changes in the ground truth model over time. For the preprocessing step, we compared the estimated posterior distributions after the clustering process to the ground truth distributions. For the adaptive process, we compared the estimated posterior distributions during the adaptive process to the known generative distributions. Three versions of the adaptive process are evaluated.

1. The full algorithm with clusters created, adapted and merged. We refer to this version as EST_FP.

[2]Code and data are available online https://github.com/annafriebe/AdaptiveETBayes.

2. The online algorithm with cluster adaptation, but without creation and merging of clusters. We refer to this version as EST_NCM.

3. The online algorithm without creating, adapting or merging clusters, only switching between the clusters resulting from the preprocessing stage. We refer to this version as EST_SP.

To evaluate the similarity between the posterior estimates and the ground truth distributions, the KL divergence is calculated. The KL divergence is an asymmetric measure of the difference from a distribution Q to another reference distribution P, with continuous probability density functions $q(x)$ and $p(x)$ respectively, defined as

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \, dx. \tag{9.31}$$

The KL divergence was chosen as it quantifies the information lost when moving from the ground truth distribution to the estimated distribution. The GLR is not suitable for this evaluation because it is based on likelihoods of observations.

The KL divergence is numerically approximated from the estimated posterior distribution to the ground truth distribution in the range $[0, 150]$. The posterior distribution is constructed by weighting the generalized student's t distributions of each state with the stationary probabilities of the states in the fitted HMM. The ground truth normal distributions are similarly weighted with the known stationary probabilities of states.

9.6.2 Generation of Sequences from the Ground Truth Model

Execution sequences are generated from the ground truth model defined as a three state Markov model, where transition probabilities between states are in the range 0.1-0.8. Each sequence is constructed from five clusters, and each cluster is constructed from the segments of length within interval $[50, 300]$. One of the clusters does not appear until after the first 1000 job indices, which means it is not in the preprocessing section. The execution time samples for each cluster and its respective segments, are generated according to a three state

Sequence	1	2	3	4
Cluster 1	0.111	0.054	0.158	0.109
Cluster 2	0.149	1.663	0.045	0.036
Cluster 3	0.051	0.323	0.081	0.580
Cluster 4	0.151	0.067	0.116	0.174
All clusters	0.107	0.156	0.085	0.107

Table 9.3: KL divergence measures for the preprocessing process.

Markov Model, such that each state is characterized by a Gaussian emission distribution with a mean randomly generated from one of the three following uniform ranges $[25, 50]$, $[65, 80]$ and $[95, 120]$ respectively and standard deviations within the range $[2, 6]$. The cluster means are ordered, so that if the mean of a state in cluster A is lower than the mean of the same state in cluster B, $\mu_{nA} < \mu_{nB}$, then the same relation applies to the other states' means in these clusters. The reason for this is that points of model change are not as accurately found when the state means of two clusters move in opposite directions. This is likely due to the construction of the GLR of segments as the sum over the states.

9.6.3 Results

Preprocessing Step

In Fig. 9.3 we show means and standard deviation of the estimate, i.e. the posterior generalized student's t distributions of the resulting clusters as black lines. We also show the means and standard deviations of the ground truth clusters as red lines. For sequence 2, four states are identified, and the state with the lowest stationary probability is displayed in cyan. In Fig. 9.3, points of model change are also visible. KL divergence measures along the sequences are displayed, in black for the preprocessing section. Mean KL divergence measures for each cluster and for the preprocessing section are displayed in Table 9.3.

Online Adaptive Process

The means and standard deviations of the posterior generalized student's t distributions during the adaptive process are displayed in blue in Fig. 9.3 for four sequences. These are shown in relation to the known means and standard deviations of the normal distributions in the true clusters in red. For sequence 2, the HMM identification finds four states, and the state with the lowest stationary probability is displayed in cyan. In Fig. 9.3 the points of model change are also visible.

The left column displays the result when applying EST_FP, the full process, to four test sequences. Creation of new clusters is indicated with black vertical lines, and merging of clusters is marked with red vertical lines. The middle column shows the result when applying EST_NCM, without creation and merging of clusters, but with adaptive cluster updates for the same sequences. The right column displays the result when applying EST_SP, with only switching between the preprocessing clusters.

The KL divergence from the distribution constructed from the posterior generalized student's t distributions to the distribution constructed from ground truth Gaussian distributions is calculated. When constructing the distributions, the emission distributions are weighted with the estimated and known stationary probabilities respectively. The KL divergence is calculated for each job index in each sequence for the three versions of the adaptive process. Results are displayed in Fig. 9.3. Means are calculated for each ground truth cluster and for the entire adaptive part of the sequence, and presented in Table 9.4. In sequences 1, 3 and 4, EST_SP has a better average fit (lower average KL divergence measure), as can be seen in the "All clusters" row of the tables. We also look at the average KL divergence of clusters not appearing in the preprocessing portion, that is Cluster 5 for all sequences, and for sequence 2 additionally Cluster 2. Here we see that EST_FP has lower KL divergence measures than EST_SP in four out of the five new clusters. For the EST_NCM, the KL divergence is lower for all five new clusters. EST_FP and EST_NCM appear to be roughly equivalent for new clusters, with the EST_NCM having lower KL divergence scores in three out of five new clusters.

Sequence no.	Cluster no.	EST_FP	EST_NCM	EST_SP
1	1	0.347	0.248	0.277
	2	0.276	0.276	0.310
	3	N/A	N/A	N/A
	4	0.770	0.786	0.442
	5	0.411	0.266	0.359
	All clusters	0.459	0.401	0.346
2	1	0.173	0.173	0.217
	2	0.263	0.261	0.681
	3	0.493	0.493	0.539
	4	0.340	0.342	0.110
	5	0.355	0.362	0.400
	All clusters	0.297	0.297	0.392
3	1	0.460	0.608	0.478
	2	0.506	0.257	0.139
	3	1.142	1.180	0.808
	4	0.285	0.282	0.159
	5	0.270	0.503	0.512
	All clusters	0.688	0.739	0.536
4	1	0.241	0.242	0.215
	2	0.347	0.281	0.046
	3	0.498	0.502	0.620
	4	0.414	0.417	0.171
	5	0.631	0.627	0.760
	All clusters	0.418	0.405	0.373

Table 9.4: KL divergence measures for different sequences and clusters.

9.6.4 Discussion

The KL divergence in the adaptive section is in the range of 2-10 times larger than in the preprocessing section in our experiments, for all three versions of the adaptive process. A larger KL divergence is expected from a less computationally expensive approach.

The fact that EST_SP performs better than the versions with cluster updates for clusters available at the preprocessing step indicates that there is some deterioration of the estimates, possibly due to erroneous estimates of the points of cluster change.

It can be noted that in some segments, the estimated means of the states with the highest and lowest means tend to move towards the middle state in the three state HMM, coinciding with a higher standard deviation. This is likely due

to samples generated by the middle state distribution resulting in occupancy probabilities significantly higher than zero for an additional state. When these samples are weighted into the sufficient statistics of the lower or higher state, the posterior distribution is affected in this manner.

The choice of prior distribution has a similar influence on the posterior estimates. In the proposed method, the prior distribution is based on the HMM fitted to the preprocessing section. For portions of the execution time trace that deviate significantly from the preprocessing section, the posterior estimates will have a mean that is drawn towards the prior mean, and a variance that is overestimated due to the prior pseudo observations acting as outliers.

9.6.5 Limitations and Future Evaluation Goals

The main limitation of the evaluation is that it has been performed with synthetic data, where the execution time samples are generated from ground-truth distributions with instantaneous cluster changes at specified points in time. The main reason for this choice was the controllable experiment setup where the ground truth model is known. One of the sensitive design choices of the experiment is evident in the generation of the ordered means within the clusters, which should be generalised in the future evaluations. Also, at the moment we cannot be certain that the results are valid for more realistic use cases and this will be addressed in the future work.

9.7 Conclusion and Future Work

In this paper, we proposed a method to adjust at runtime an HMM aimed at characterizing the execution time of a task, with a limited time complexity. The posterior execution-time distributions obtained through the proposed approach could be used to assess several real-time properties of a system, e.g., estimating the deadline miss probabilities, but further investigations are needed, and devoted to future work.

The results from the evaluated synthetic test cases indicate that the proposed method is capable of adapting the estimates at runtime, such that the estimated distribution tracks the ground truth distribution used to generate the

execution time samples. The similarity between the estimated and ground truth distributions are evaluated by calculating the Kullback-Leibler divergence. In some cases we can see biased means and increasing standard deviations in the posterior distribution. Future work will investigate the possibility of introducing regularization to limit the increase in the standard deviation. Furthermore, a more extensive evaluation will be performed on real applications, e.g., computer vision, robotics, or control use cases, to better assess the ability of the proposed approach to provide meaningful information on the execution time distributions of complex real-time applications.

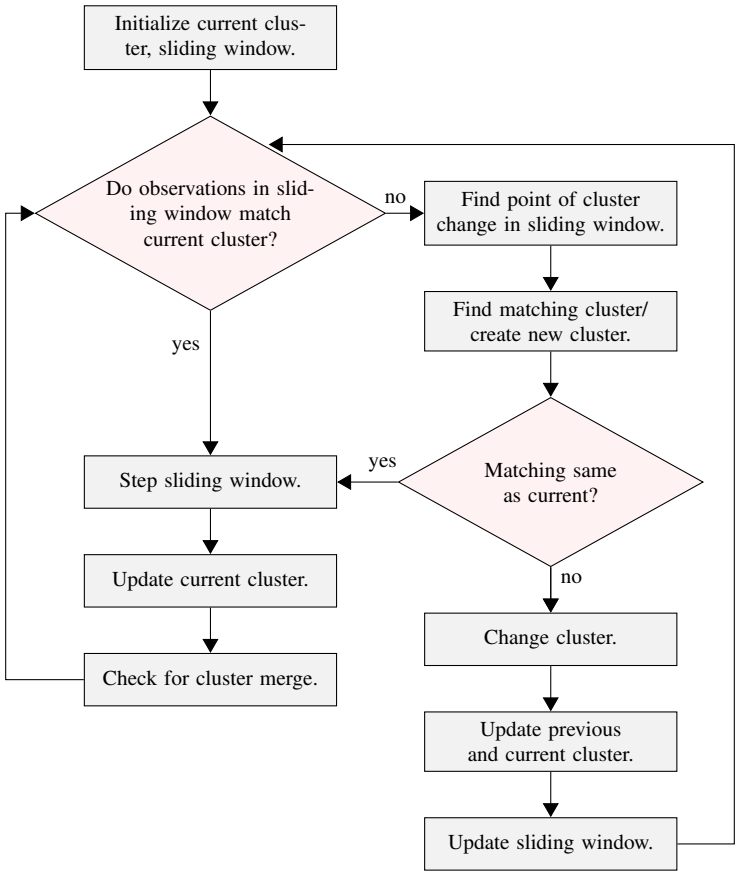


Fig. 9.2: Simplified flowchart of the adaptive process. The process continues until the task is terminated and there are no more observations to process.

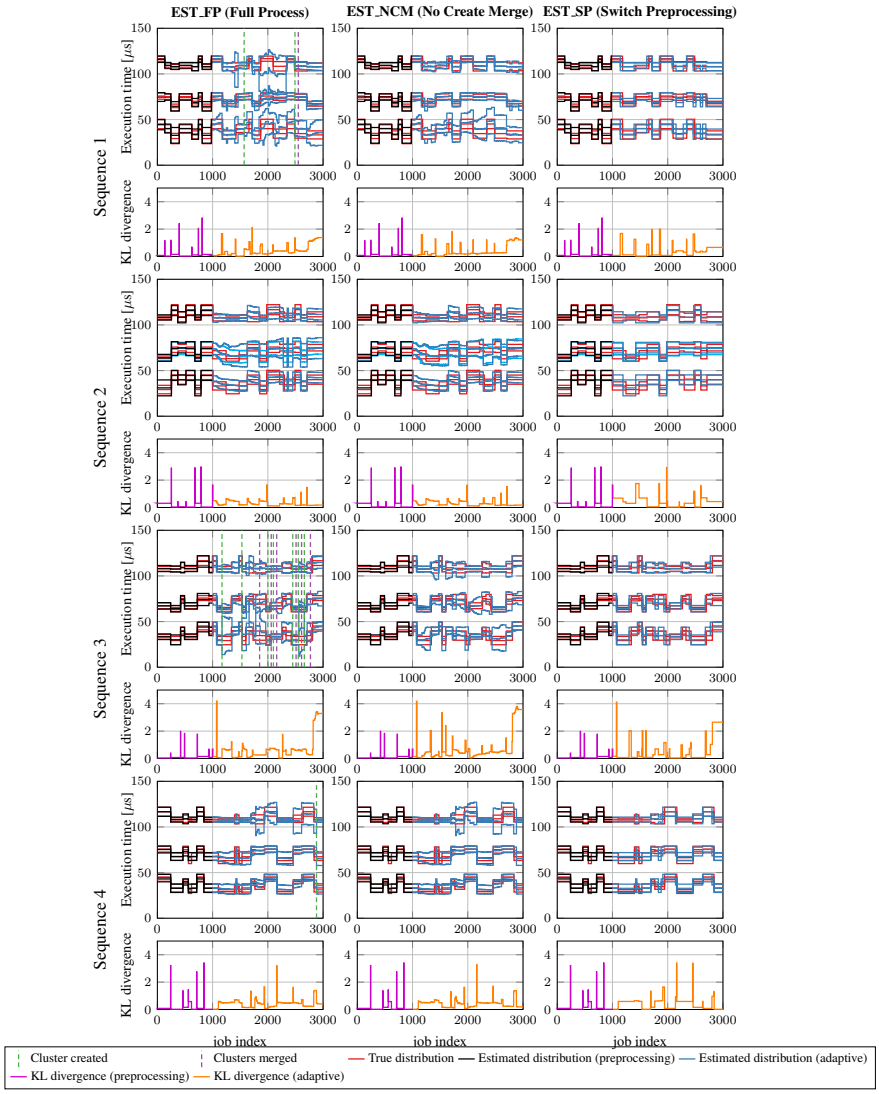


Fig. 9.3: True and predicted distributions for four sequences, with the three different versions of the process. KL divergence measures along the sequences are displayed.

Bibliography

[1] Luca Abeni, Daniele Fontanelli, Luigi Palopoli, and Bernardo Villalba Frías. A Markovian model for the computation time of real-time applications. In *IEEE international instrumentation and measurement technology conference (I2MTC)*, pages 1–6, 2017.

[2] Christopher M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Alan Burns and Robert I. Davis. A survey of research into Mixed Criticality Systems. *ACM Computing Surveys (CSUR)*, 50(6), 2017.

[4] Alan Burns and Stewart Edgar. Predicting computation time for advanced processor architectures. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 89–96, 2000.

[5] Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Computing Surveys (CSUR)*, 52(1), 2019.

[6] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *SIGCOMM Comput. Commun. Rev.*, 22(4):14–26, 1992.

[7] Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J Cazorla. Measurement-Based Proba-

bilistic Timing Analysis for multi-path programs. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 91–101, 2012.

[8] Robert Ian Davis and Liliana Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for Real-Time systems. *LITES: Leibniz Transactions on Embedded Systems*, pages 1–53, 2019.

[9] Robert Ian Davis and Liliana Cucu-Grosjean. A survey of probabilistic timing analysis techniques for Real-Time systems. *LITES: Leibniz Transactions on Embedded Systems*, 6(1):03–1–03:60, 2019.

[10] Jose Luis Diaz, Jose Maria Lopez, Manuel Garcia, Antonio M Campos, Kanghee Kim, and Lucia Lo Bello. Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 197–207, 2004.

[11] Richard O Duda, Peter E Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

[12] Stewart Edgar and Alan Burns. Statistical analysis of WCET for scheduling. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 215–224, 2001.

[13] Bernardo Villalba Frias, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. Probabilistic real-time guarantees: There is life beyond the i.i.d. assumption. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 175–186, 2017.

[14] Anna Friebe, Alessandro V. Papadopoulos, and Thomas Nolte. Identification and validation of Markov models with continuous emission distributions for execution times. In *IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10, 2020.

[15] David Griffin and Alan Burns. Realism in statistical analysis of Worst Case Execution Times. In *Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, 2010.

[16] Christian Gruhl, Jörn Schmeißing, Sven Tomforde, and Bernhard Sick. Normal-Wishart clustering for novelty detection. In *IEEE International*

Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), pages 64–69. IEEE, 2020.

[17] Mathai Joseph and Paritosh Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[18] M Ross Leadbetter, Georg Lindgren, and Holger Rootzén. Conditions for the convergence in distribution of maxima of stationary normal processes. *Stochastic Processes and their Applications*, 8(2):131–139, 1978.

[19] D Liu and Francis Kubala. Online speaker clustering. In *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, volume 1, pages 333–336, 2004.

[20] Andreas Löfwenmark and Simin Nadjm-Tehrani. Fault and timing analysis in critical multi-core systems: A survey with an avionics perspective. *Journal of Systems Architecture*, 87:1–11, 2018.

[21] Chenyang Lu, John A Stankovic, Sang H Son, and Gang Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1):85–126, 2002.

[22] Yue Lu, Thomas Nolte, Iain Bate, and Liliana Cucu-Grosjean. A statistical response-time analysis of complex real-time embedded systems by using timing traces. In *IEEE Int. Symp. on Industrial and Embedded Systems (SIES)*, pages 43–46, 2011.

[23] Yue Lu, Thomas Nolte, Iain Bate, and Liliana Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 351–362, 2012.

[24] Yue Lu, Thomas Nolte, Johan Kraft, and Christer Norstrom. A statistical approach to response-time analysis of complex embedded real-time systems. In *IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 153–160, 2010.

[25] Yue Lu, Thomas Nolte, Johan Kraft, and Christer Norstrom. Statistical-based response-time analysis of systems with execution dependencies between tasks. In *IEEE Int. Conf. on Engineering of Complex Computer Systems*, pages 169–179, 2010.

[26] Pau Martí, Josep M Fuertes, Gerhard Fohler, and Krithi Ramamritham. Improving Quality-of-Control using flexible timing constraints: metric and scheduling. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 91–100, 2002.

[27] Todd K. Moon. The Expectation-Maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[28] Kevin P. Murphy. Conjugate Bayesian analysis of the Gaussian distribution. Technical report, University of British Columbia, 2007.

[29] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. the IEEE*, 77(2):257–286, 1989.

[30] Luca Santinelli, Jérôme Morio, Guillaume Dufour, and Damien Jacquemart. On the sustainability of the extreme value theory for WCET estimation. In *Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, 2014.

[31] Takahiro Shinozaki. HMM state clustering based on efficient cross-validation. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, 2006.

[32] Reinhard Wilhelm. Mixed Feelings About Mixed Criticality. In *Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 63, pages 1:1–1:9, 2018.

[33] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al. The Worst-Case Execution-Time problem—overview of methods and survey of tools. *ACM TECS*, 7(3):36, 2008.

Chapter 10

Paper C
Efficiently Bounding Deadline Miss Probabilities of Markov Chain Real-Time Tasks.

Anna Friebe, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte.
In Real-Time Systems, 2024.

Abstract

In real-time systems analysis, probabilistic models, particularly Markov chains, have proven effective for tasks with dependent executions. This paper improves upon an approach utilizing Gaussian emission distributions within a Markov task execution model that analyzes bounds on deadline miss probabilities for tasks in a reservation-based server. Our method distinctly addresses the issue of runtime complexity, prevalent in existing methods, by employing a state merging technique. This not only maintains computational efficiency but also retains the accuracy of the deadline-miss probability estimations to a significant degree. The efficacy of this approach is demonstrated through the timing behavior analysis of a Kalman filter controlling a Furuta pendulum, comparing the derived deadline miss probability bounds against various benchmarks, including real-time Linux server metrics. Our results confirm that the proposed method effectively upper-bounds the actual deadline miss probabilities, showcasing a significant improvement in computational efficiency without significantly sacrificing accuracy.

10.1 Introduction

Soft real-time systems permit limited deadline misses, impacting the Quality of Service (QoS) [14] or Quality of Control (QoC) [47] to a tolerable degree. The tolerance is often modeled as a constraint on the number of deadline misses to maintain an acceptable level of QoS or QoC [10]. Findings from a recent survey by Åkesson *et al.* [6] indicate a predominant presence of soft real-time systems in the industry, underscoring the significance of their analytical study.

Hidden Markov Models (HMMs) have been effectively utilized to model execution times in systems with dependencies that exhibit regular variations. An introduction to the HMM concept can be found in [51]. Studies like [4, 25] have employed Markov models with discrete emission distributions, particularly in estimating the probability of missing deadlines under a Constant Bandwidth Server (CBS). Additionally, continuous-emission distributions have been explored by Friebe *et al.* [29, 27, 28].

While HMMs with continuous emission distributions have been applied in execution time estimation [29, 27], the extension to workload distribution inference and deadline-miss probabilities has been a recent development [28]. As in previous work [4, 25], this analysis is done with a reservation-based scheduling approach, that allows for analysis of each server separately, due to the timing isolation property. This newer exploration has shown potential but highlighted challenges in computational efficiency when dealing with complex systems.

Building upon these recent findings, our paper specifically targets the computational efficiency issue identified by Friebe *et al.* [28]. We propose an enhanced method for bounding the deadline-miss probability of real-time tasks using HMMs with continuous emission distributions. A key contribution of this work is developing a state-merging technique that enhances computational efficiency in terms of time and space complexity, where traditional methods are computationally intensive or even infeasible (Section 10.6).

The evaluation, presented in Section 10.7, employs a task managing a Furuta pendulum [57]. It compares the derived bounds with real-time deadline miss ratios under Linux's CBS implementation, `SCHED_DEADLINE` [34], alongside estimates from a discrete-emission Markov Model [25, 26], and

simulation-based estimates.

The paper's organization is as follows: Related work is reviewed in Section 10.2. Section 10.3 defines the notation and system model. The execution time model and methodology for deriving and analyzing the deadline miss probability bounds are presented in Section 10.4. The iterative update process for the bounds is detailed in Section 10.5. State reduction techniques and their impact on time complexity are discussed in Section 10.6. Section 10.7 showcases evaluations and results, and Section 10.8 concludes the paper with a discussion on future work.

10.2 Related Work

The surveys by Davis and Cucu-Grosjean [20, 19] offer a detailed overview of the field of probabilistic schedulability and timing analysis in real-time systems. Two of the many challenges highlighted in their surveys are the probabilistic analysis of dependent tasks and the safe estimation of deadline-miss probabilities (DMP) for such tasks.

The issue of dependence in execution-time distributions and its impact on the potential unsound estimation of DMP when independence is assumed was initially identified by Tia *et al.* [54]. The importance of assuming independence among jobs of the same task for deriving sound response time distribution was first recognized by Diaz *et al.* [22], while in the concluding remarks of their paper, they restated the fact that many systems do not adhere to the independence assumption. For this reason, a fundamental concept of *stochastic pessimism* for proper upper-bounding of the execution-time distributions was explored by Diaz *et al.* [23]. Over the years, several research directions have evolved to address the above-mentioned issues.

One of the most used approaches was the one based on the probabilistic Worst-Case Execution Time (pWCET), which is supposed to upper-bound the ground-truth execution-time distribution of a job such that it can be safely used with convolution and independence-assuming analytical approaches in spite of possible dependence with other jobs. In this line of research, Cucu-Grosjean *et al.* [16] established the relation between the ground-truth execution-time distributions and pWCET, while Davis *et al.* [18] clarified the difference between the

confidence-based pWCET and the upper-bounding one. The surveys [20, 19] also provide extensive investigation on the definition and use of pWCETs. More recently, Bozhko *et al.* [8] formalized a rigorous, axiomatic definition of pWCET, using the Coq proof assistant.

Many probabilistic schedulability analyses have been proposed over the years using pWCET and similar independence-assuming distributions for fixed-priority fully-preemptive scheduling. Von der Brüggen *et al.* [55] used the Hoeffding and Bernstein inequalities for the estimation of DMP, while Chen *et al.* [13] used Chernoff bound. Marković *et al.* [44] contributed an optimal resampling strategy and an efficient circular-convolution algorithm. Bozhko *et al.* [9] introduced a method based on Monte-Carlo sampling. In contrast, von der Brüggen *et al.* [56] suggested a method to approximate the DMP under Earliest Deadline First (EDF) scheduling, accommodating dependencies across a limited number of consecutive jobs. More recent work by Chen*et al.* [12] corrected an error in the critical-instant assumption commonly found in various independence-based methods. Zagalo *et al.* [58] have developed queuing theory-based approximations for the response-time distributions, while Marković *et al.* [46] utilized the Berry-Esseen theorem to approximate response-time distributions. In the context of other scheduler assumptions, Marković *et al.* [43] provided probabilistic analysis for limited-preemptive scheduling, which is a generalization over fully and non-preemptive scheduling. Most recently, Günzel *et al.* [30] proposed a probabilistic reaction time analysis for cause-effect chains based on sporadic tasks.

The issue of dependence has also been addressed within the framework of Extreme Value Theory (EVT), particularly in its application to measurement-based statistical analysis for both execution times [17, 36, 35] and response times [40, 41, 39].

Despite the widespread adoption of EVT in both academic research and practical applications, it is not without certain limitations [21]. EVT relies on the premise that statistical limit laws are applicable to the sample set at hand [15]. EVT necessitates certain conditions like the assumption of stationarity [33] or extremal independence in the distribution under consideration [52].

Regarding the works that do not consider independence-assuming distributions, Bernat *et al.* [7] introduced the concept of copulas in timing analy-

sis. Copulas model dependencies between random variables, a copula transforms marginal distributions of random variables into a joint probability distribution. Ivers and Ernst[32] developed an approach for fixed-priority preemptive scheduling systems, utilizing completely known ground-truth probability distribution for each task. Their method, incorporating copulas and Frechet bounds, facilitated the derivation of probabilistic response-time bounds. Recently, Marković *et al.* [45] introduced a correlation-tolerant analysis for DMP estimation, leveraging upper bounds on both the expected values and standard deviations of job execution-time distributions. Their analysis improves upon Cantelli's inequality to derive sound probabilistic response times in the presence of possibly correlated distributions.

More in line with this work, in the context of *server-based schedulers*, Mills *et al.* [49] derived bounds for response time and tardiness for soft real-time tasks with stochastic execution times, focusing on execution time dependence within distinct time windows. In a related development Liu *et al.* [37] proposed the concept of independence thresholds, positing that execution times above a certain threshold can be treated as independent. One major advantage of server-based scheduling is that it provides timing isolation, allowing for analysis of each server separately.

The *Constant-Bandwith Server* (CBS), was originally introduced by Abeni *et al.* [1] and later used for deriving probabilistic deadlines to ensure Quality of Service (QoS) guarantees [2]. In later works, it has also been analyzed with probabilistic execution times[3, 50] and probabilistic interarrival times [5, 42].

In one of the seminal papers for probabilistic analysis of real-time systems, Diaz *et al.* [22] conducted a response time analysis for periodic tasks characterized by independent random execution times, demonstrating that the backlog in this context can be modeled as a *Markov chain*.

Recent studies, diverging from the worst-case DMP that has been prevalent in the previously cited works, have embraced the long-run frequency interpretation of DMP. In this vein, [25, 4] utilized Markov chain models with discrete emission distributions under CBS. Their work concentrated on analyzing the steady-state response time distribution and included comparisons with results obtained under Linux's `SCHED_DEADLINE`. They noted that the analysis dura-

tion is influenced by factors such as the range of execution times, the number of states, and the scaling factor used for resampling [24], which can significantly affect the analysis time and space complexity.

Furthermore, the estimation of the execution times modeled as continuous Gaussian distributions within Markov chains have been explored by Friebe *et al.* [29, 27], while the analysis in terms of deadline miss probabilities was conducted recently [28]. Friebe *et al.* [28] addressed the DMP estimation, applying Hidden Markov Models (HMMs) with Gaussian emission distributions for schedulability analysis. This approach, akin to the work of Frías *et al.* [25] and Abeni *et al.* [4], explicitly incorporates dependencies within the HMM framework, with the CBS providing task isolation, thereby focusing the workload analysis on the specific task rather than the entire system. Although the analysis by Friebe *et al.* [28] offered improvements due to utilizing the continuous-based HMM model of execution times, they showed that the analysis still may suffer from time and space complexity.

In Section 10.6 of this paper, we introduce state-merging techniques designed to enhance the time and space efficiency of the methods presented by Friebe *et al.* [28]. These techniques are developed to maintain high accuracy in DMP estimations. In Table 10.1 the HMM approaches with continuous and discrete emission distributions are compared, and the contributions of this paper are outlined.

10.3 System Model and Notation

Table 10.2 contains the notation used in the paper. Superscript * indicates true values, $^\uparrow$, and $^\downarrow$ indicate upper and lower bounds.

We use the concept of upper bounding random variables according to Definition 10.3.1. This is also referred to as the usual stochastic order [53] or first-order statistical dominance [23]. This paper uses the term upper bound as in [19].

Definition 10.3.1 (cf. [23, 19, 53])**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be two random variables. We say that $\mathcal{X}$ is greater than or equal to $\mathcal{Y}$ (i.e., $\mathcal{X}$ upper bounds $\mathcal{Y}$) if the Cumulative Distribution Function (CDF) of $\mathcal{X}$ is never above that of $\mathcal{Y}$. We denote this relation by $\mathcal{X} \geq \mathcal{Y}$.*

Table 10.1: Contribution overview.

	Discrete Emission	**Continuous Emission**
Overview of the theoretical contributions		
Timing Analysis (TA)	[25] [4]	[29]
DMP analysis	[25] [4]	[28] as described in this paper
Comparison of the models and their analysis properties		
State No. identification	–	[29]
Adaptive TA	–	[27]
Time and space complexity (DMP analysis)	[25] [4] **Dependent on:** - resampling scale - execution time range Requires full steady-state distribution [25].	[28] as described in this paper **Independent of:** - resampling scale - execution time range Iterative procedure with an adjustable complexity. State number reduction procedure (Section 10.6).

We define a partial Gaussian distribution in Definition 10.3.2, that is used to upper bound workload distributions. Consider a Gaussian $\mathcal{N}(\mu, \sigma^2)$ with probability density function $f(x|\mu, \sigma^2)$. Let $\Phi(x)$ be the cumulative density function of the standard normal distribution.

Definition 10.3.2. *A partial Gaussian distribution $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha)$, originated from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, is defined by the probability density function:*

$$f^{tail}(x|\mu, \sigma^2, \alpha) = \begin{cases} 0, & x \leq \alpha \\ \frac{1}{\Phi(\frac{\mu-\alpha}{\sigma})} \cdot f(x|\mu, \sigma^2) & x > \alpha \end{cases}. \tag{10.1}$$

The probability of values lower than α is set to zero in the partial Gaussian distribution. The probability density of the remaining values are normalized, so that the distribution integrates to one.

We use convolutions, as defined in Definition 10.3.3, in the derivation of workload distributions.

Definition 10.3.3. *The convolution of f and g, denoted with the $*$ operator is:*

$$[f * g](z) = \int_{-\infty}^{\infty} f(z - x) \cdot g(x) \, \mathrm{d}x.$$

10.3.1 Task Model

Let the real-time task τ consist of a sequence of jobs J_i, $i \in \mathbb{N}$. Each job J_i has the arrival time a_i, execution time c_i and finishing time f_i. The task is periodic and jitter-free, i.e., $a_{i+1} = a_i + T$, a_0 is the arrival time of the first job. Jobs can be preempted, $f_i \geq a_i + c_i$. The execution time is modeled as a random variable. The random variable $\mathcal{R}$ models the response time, the duration from activation time to finish time of a job.

A job J_i has the deadline d_i determined by a relative deadline D such that $d_i = a_i + D$. Jobs are executed until completion, even if deadlines are missed. The relative deadline may be longer than the task period. The probability that a randomly selected job finishes after the deadline, $p_{dm} = \mathbb{P}(\mathcal{R} > D)$ is the main concern of this paper.

10.3.2 Scheduling Algorithm

The task is served as the sole task of a reservation-based server, and guaranteed to receive Q units of processing time within each server period. The bandwidth $B = Q/P$ is the fraction of the processing time dedicated to the task. $T = n \cdot P$, that is the task period is an integer multiple of the server period. The relative deadline is also an integer multiple of the server period, $D = k \cdot P$. In the evaluation a CBS is used, a CBS with a properly selected server period fulfills the necessary requirements.

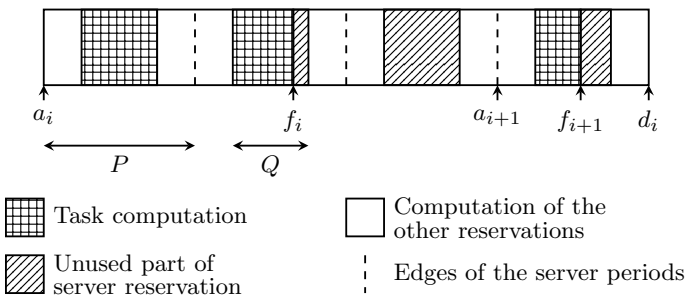


Fig. 10.1: An illustration of the task model and the reservation-based server.

The task model and reservation-based server are illustrated in Fig. 10.1. Here, we have $n = 3$, $T = 3 \cdot P$. In the illustration the relative deadline is longer than the period. With a deadline longer than the period, a longer job may steal computation time from the next job of the task. If the next job is short, they may both meet their deadlines.

10.4 Execution Time Model and Analysis

10.4.1 Markov Chain Execution Times

The execution times of the task we consider are described by a Markov model defined by a set of S states $\mathbb{S}$, a $S \times S$ state transition matrix M and a set $\mathbb{C}$ of S execution time distributions or *emission distributions* related to the respective state, $S \in \mathbb{N}$. We have $\mathbb{S} = \{1, 2, \ldots, S\}$. In M the element $m_{a,b}$ represents the probability of the task being in state b at task period $i + 1$, given that it is in state a at task period i. $\mathbb{C} = \{C_1, C_2, \ldots, C_S\}$ where each C_s is modeled as Gaussian distributions with mean μ_s, and variance σ_s^2, i.e., $C_s \sim \mathcal{N}(\mu_s, \sigma_s^2)$. The Markov Chain is irreducible, that is a chain where from any state you can reach any other state in a sequence of steps. For an irreducible finite-state Markov Chain, stationary probabilities of the different states [31] exist and are unique. The stationary probabilities represent the long-run proportion of jobs with execution times described by the different Gaussian distributions.

Example 10.4.1. *When introducing the ideas and analysis, we will use an ex-*

ample execution time Markov Model and reservation-based server. The parameters are chosen mainly for illustration, and to arrive at simple numerical answers in some of the applications of the example. The Markov Model is defined by:

$$S = 2, \qquad M = \begin{pmatrix} 0.9 & 0.1 \\ 0.7 & 0.3 \end{pmatrix}, \qquad \mathcal{C} = \{\mathcal{N}(1, 0.25), \mathcal{N}(2, 1)\}.$$

In this example all transition probabilities are strictly positive, so the Markov Chain is clearly irreducible and we can calculate the stationary probabilities. These are 0.875 *for state* 1 *and* 0.125 *for state* 2*. In our example, the CBS is defined as* $n = 2$ *and* $Q = 1$*. The deadline is defined by* $k = 4$*.*

The representation of Gaussian emission distributions requires only a few distribution parameters, for example, the emission distribution associated with state 1 in Example 10.4.1 is fully specified by the mean 1 and the variance 0.25. With discrete distributions probabilities for each execution time value need to be stored, with respect to a chosen scaling factor. For Example 10.4.1 we might choose to represent execution times with a resolution of 0.01. For state 1 we could list execution times from 0.01 to 4.01, where each of these is associated with a probability. The probability associated with 2.01 would be the probability of execution times et, $2 < et \leq 2.01$. Gaussian emission Markov models are shown to be applicable, in Friebe *et al.* [29] for a video decompression use case, and in Friebe *et al.* [27] in a dynamic setting with model adaptation. The Gaussian distribution may appear simplistic. However, general distribution shapes can be approximated by a combination of Gaussians. The assumption of independent execution times within each state implies that more states may be necessary in the model to capture dependencies in the transition matrix. A close to discrete model can be envisioned with states where the emission-distribution variance is near zero. The fact that a representation with Gaussian emission distributions may require more states is a disadvantage of this approach. With discrete representation, pessimism is introduced with the scaling factor and if downsampling is needed. With continuous representation, pessimism is introduced at other points, for example in our case when upper bounding the workload distributions. Further, there are other options for

continuous representations, for example Zagalo *et al.* [58] use inverse Gaussian mixture distributions for response times. In our approach we rely on the simplicity of convolution of Gaussian distributions.

10.4.2 Problem Formulation

We bound the expected deadline miss probability of a randomly selected job of a task. Task execution times are defined as in Section 10.4.1, and the task is served by a reservation-based server as described in Section 10.3.2.

In the survey on schedulability analysis by Davis and Cucu-Grojean [20] three interpretations of the probability of a deadline miss are listed:

1. "As a probability with a long-run frequency interpretation equating to the expected number of missed deadlines divided by the total number of deadlines in a long (tending to infinite) time interval.

2. As the probability that a randomly selected job will miss its deadline, which is broadly equivalent to the long-run frequency interpretation.

3. As a bound on the probability that any specific job will miss its deadline."

Chen *et al.* [11] refer to the same concept as the deadline miss rate, and formulate the question: "What is the ratio of jobs missing their deadlines in the long run?" We agree with Davis and Cucu-Grosjean that interpretations 1 and 2 are broadly equivalent. Extending interpretation 2 to include the average component that is in focus in interpretation 1, we focus on the expected deadline miss probability of a randomly selected job. The intention is to remove any ambiguity with interpretation 3 or the Worst Case Deadline Failure Probability, an upper bound on the probability that any single job of a task misses its deadline [20]. We find the term deadline miss probability more natural compared to deadline miss rate in our context with states with different execution time distributions. .

10.4.3 Overview of the Proposed Approach

We will obtain an upper bound on the expected deadline miss probability of a randomly selected job of the task in a reservation-based server.

The proposed method is based on a workload accumulation scheme. The main idea is outlined below, followed by the details in the remaining subsections. The starting point of the approach is that the deadline miss probability of a job depends on the execution time of the job, and on the amount of remaining work from previous jobs that have not been completed yet. We categorize jobs into different classes with different deadline miss probabilities. By calculating or bounding the deadline miss probabilities of jobs belonging to each class, and the probability of randomly selecting a job from each class, we bound the expected deadline miss probability of a randomly selected job.

In each task period, task τ is guaranteed nQ units of processing time. The pending workload at the i-th task period is denoted as v_i and defined as in Abeni *et al.* [2]:

$$v_i = \underbrace{\max(0, v_{i-1} - nQ)}_{\text{carry-in workload}} + c_i. \tag{10.2}$$

where the first term accounts for the previous workload, that is 0 for the first period, and for task periods where all work from previous jobs has been completed before the new job arrival. In these periods, jobs arrive at idle points with 0 carry-in workload and $v_i = c_i$, in particular $v_1 = c_1$.

Observation 10.4.1. *The pending workload at a job arrival is affected by the execution time requirements of jobs arriving since the last idle point.*

In our proposed method, the job classes are related to the state sequence since the last idle point. In Example 10.4.1, let the jobs arriving at an idle point when the task is in the first state of the Markov Model belong to one specific class. When selecting a job at random, there is a probability of about 0.78 that the job belongs to this class. The deadline miss probability for this class of jobs only depends on the execution time distribution for the first state and the server properties. It is the survival function or 1-CDF of $\mathcal{N}(1, 0.25)$ at 4, about $9.8 \cdot 10^{-10}$.

Observation 10.4.2. *The deadline miss probability for a class of jobs is at most 1.*

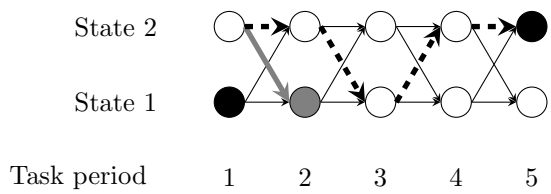


Fig. 10.2: Illustration of workload accumulation sequences of the example.

In the proposed method, we will derive more precise bounds for several classes. For the remaining, we will use Observation 10.4.2 to upper bound the deadline miss probability. We construct an approximate bound of the expected deadline miss probability of a randomly selected job from Example 10.4.1 to illustrate the idea:

$$DMP \lessapprox 0.78 \cdot 0.98 \cdot 10^{-10} + 0.22 \cdot 1 \tag{10.3}$$

To model a state sequence from the latest idle point, we introduce the concept of workload accumulation sequences. Illustrations of workload accumulation sequences for some classes in Example 10.4.1 are displayed in Fig. 10.2.

The class where, at a job arrival, the task is in state 1, and there is no carry-in workload, is displayed as the black node at task period 1. The workload accumulation sequence is $h = (1)$. The gray node and arrow represent the class of jobs where jobs arrive in state 1 and in the second task period after an idle point, with the first period in state 2. The accumulation sequence $h = (2, 1)$. The black node and dashed arrows represent the class of jobs arriving at the fifth task period after the latest idle point, with $h = (2, 2, 1, 2, 2)$. The *accumulation sequence* is modeled as a random variable $\mathcal{H}$ that can take the values of any possible workload accumulation path. In Fig. 10.2, with the gray path we illustrate one possible value $h = (2, 1)$, taken by $\mathcal{H}$.

Definition 10.4.1. *Each arrival of a job J_i results in an* accumulation sequence $h(J_i)$. *Let b denote the task state at the arrival of J_i. If there is an idle point directly prior to the arrival, the resulting $h(J_i) = (b)$. If there is carry-over workload from the previous job, let $h(J_{i-1}) = (\ldots, a)$ denote the accumulation sequence resulting from the prior job arrival. Then $h(J_i) = (\ldots, a, b)$.*

In this way, each job that arrives is related to one specific h that describes the accumulated workload since the last idle point, and the task's state at the arrival of this job is always in the last component of the corresponding h.

The evolution of $\mathcal{H}$ is described by an infinite-state Discrete-Time Markov Chain, and each workload accumulation sequence represents one job class and one state in this chain. State transitions occur at job arrivals. The possible transitions from a state $h = (\ldots, a)$ are:

1. A transition from h to $(\ldots, a, b)$ has strictly positive probability if $m_{a,b} > 0$.
2. A transition from h to (b) has strictly positive probability if $m_{a,b} > 0$.
3. No other transitions from h are possible.

The probability of randomly selecting a job resulting in a certain accumulation sequence is the stationary probability of the state in the Markov Chain. This stationary probability exists for an infinite-state Discrete-Time Markov if the chain is ergodic [31] - that is when the chain is irreducible, aperiodic and recurrent. The accumulation sequence Markov Chain is irreducible if the execution time Markov Chain is irreducible. If all states can be reached from all states in the execution time Markov Chain, the same is true for the accumulation sequence Markov Chain. The accumulation sequence Markov Chain is also aperiodic, which means that the greatest common divisor of the set of integers n, such that you can get from one state to the same state in n steps, is 1 for all states. In an infinite-state Markov Chain, either all states are recurrent, and the chain is recurrent, or all states are transient. A state is recurrent if when we start in that state, the probability is 1 that we ever return to the same state. The workload in the server can be seen as a queue, and a queue is in steady-state if the average arrival rate is lower than the average service rate [48]. This is equivalent to the average utilization of the task being lower than the server's bandwidth. Under this condition the accumulation sequence Markov Chain is recurrent. Returning to Example 10.4.1, the average computation requirement over a task period is $0.875 \cdot 1 + 0.125 \cdot 2 = 1.125$, resulting in an average utilization of $\frac{1.125}{2 \cdot P} = \frac{0.5625}{P}$. Since the server's bandwidth is $\frac{1}{P}$, the task's computational requirement is met over time, and the accumulation sequence Markov Chain is ergodic.

Definition 10.4.2. *The probability $p_{in}(h)$ of randomly selecting a job resulting in the accumulation sequence h is the stationary probability of this state in the accumulation sequence Markov Chain.*

Definition 10.4.3. *The conditional probability that a job resulting in h misses its deadline is defined as $p_{dm}(h) = \mathbb{P}(\mathcal{R} > D|\mathcal{H} = h)$.*

Definition 10.4.4. *The Deadline Miss Probability DMP(j) for the j-th job since the last depletion point is defined as*

$$DMP(j) = \frac{1}{\sum_{\forall h \in H(j)} p_{in}(h)} \sum_{\forall h \in H(j)} p_{in}(h) \cdot p_{dm}(h) \tag{10.4}$$

where the set $H(j)$ represents accumulation sequences resulting from job arrivals at the j-th task period from the last idle point.

Returning to Example 10.4.1, there are two accumulation sequences in $H(1)$, arriving at an idle point. We already discussed $h = (1)$. The second is $h = (2)$, and the probability of randomly selecting a job resulting in $h = (2)$ is about $p_{in}((2)) \approx 0.099$. The deadline miss probability $p_{dm}((2))$ is the survival function of $\mathcal{N}(2, 1)$ at 4, about $p_{dm}((2)) \approx 0.023$. In our example, $DMP(1) \approx (7.6 \cdot 10^{-10} + 0.099 \cdot 0.023)/(0.78 + 0.099) \approx 2.6 \cdot 10^{-3}$.

Definition 10.4.5. *The Deadline Miss Probability DMP is the expected deadline miss probability of a randomly selected job from the task. Given a task with execution times described by the model in Section 10.4.1, and served by a reservation-based server with a bandwidth exceeding the average task utilization, DMP is obtained as*

$$DMP = \sum_{i=1}^{\infty} DMP(i) \sum_{\forall h \in H(i)} p_{in}(h). \tag{10.5}$$

Since $p_{in}(h)$ are the stationary probabilities of the accumulation sequence Markov Chain (Definition 10.4.2), the sum of $p_{in}(h)$ over all h equals 1.
Problem: *The sum of Eq.* (10.5) *has a countably infinite number of terms. This paper investigates finding a bound for DMP with a finite number of terms.*

Observation 10.4.3. *Consider a job that arrives with a carry-in workload, i.e., it does not arrive directly after an idle point. Then this job arrives $j+1$ task periods after the last idle point, and it succeeds a job arriving j task periods after the last idle point. The probability of randomly selecting a job with workload accumulation from $j+1$ periods is never higher than the probability of randomly selecting a job arriving with workload accumulation from j periods.*

The main idea is to find tighter bounds $\mathit{DMP}(i)$ for the first terms in Eq. (10.5), since due to Observation 10.4.3 the weighting sums over p_{in} are highest for the first terms. For larger i, when the sums over p_{in} are small, we let $\mathit{DMP}(i) = 1$.

Returning to Example 10.4.1, using the information from the first task period, we have:

$$\begin{aligned} \mathit{DMP} &= \sum_{i=1}^{\infty} \mathit{DMP}(i) \sum_{\forall h \in H(i)} \approx 2.3 \cdot 10^{-4} + \sum_{i=2}^{\infty} \mathit{DMP}(i) \sum_{\forall h \in H(i)} p_{in}(h) \\ &\leq 2.3 \cdot 10^{-4} + \sum_{i=2}^{\infty} \sum_{\forall h \in H(i)} p_{in}(h) \approx 0.12 \end{aligned} \tag{10.6}$$

The bound is still very pessimistic. However, going from one accumulation sequence $h = (1)$ in Eq. (10.3) to two $h = (1)$ and $h = (2)$ reduces the bound from 0.22 to 0.12.

Outline of the remainder of this section

An upper bound on *DMP* is obtained by deriving upper bounds on p_{in} and p_{dm}. The probability $p_{in}(h)$ of randomly selecting a job with the accumulation sequence h depends on the execution time distributions along h, the transition probabilities, and the probability of workload depletion in each state. The conditional deadline miss probability $p_{dm}(h)$ for jobs where the arrival results in h depend on the execution time distributions in h. We divide the workload accumulation process in two steps. We first compute upper bounds on p_{in} and p_{dm} up until N task periods from the latest idle point. As N grows, the sum of the products of $p_{in}(h)$ and $p_{dm}(h)$ approaches the true deadline miss probability. Second, the sum of p_{in} values in the remaining accumulation sequences

of length $N + 1$ to infinity, is upper bounded. We refer to this sum as β. We assume that the p_{dm} is 1 for jobs that result in these accumulation sequences, and this gives a safe upper bound on *DMP* in Eq. (10.39).

The **steps for deriving a safe bound** on *DMP* are outlined in the following sections:

Section 10.4.4: Upper bounding p_{dm} and p_{in} for the terms in Eq. (10.5) requires upper bounding the pending workload distributions associated with each accumulation sequence. In this section we describe how to derive the parameters of an upper bounding partial Gaussian distribution as given in Eqs. (10.18) to (10.20). The bounds on p_{in} also rely on a lower bound on the pending workload distributions. The parameters for a lower bounding Gaussian distribution are derived as Eqs. (10.18) and (10.19).

Section 10.4.5: For jobs arriving at an idle point, p_{in} depends on the probability of workload depletion p_{wd} for each state, the stationary state probabilities and the transition probabilities $m_{a,b}$. Upper and lower bounds are provided in Eqs. (10.22) and (10.23). Jobs arriving with carry-in workload have p_{in} depending on transition probabilities, and the probability of jobs with shorter accumulation sequences resulting in carry-over workload. We find that all p_{in} bounds are linear combinations of p_{wd} for the different states, and given in in Eqs. (10.26) and (10.27).

Section 10.4.6: Bounds on p_{wd} are derived, relying on stationary probabilities and the sum of p_{in} in accumulation periods after N, denoted as β.

Section 10.4.7: A bound on β is derived and given in Eq. (10.36). This bound is utilized for computing the lower bounds on p_{in}, p_{co}, and finally p_{wd}.

Section 10.4.8: An upper bound on p_{dm} is presented, using the bounds on workload distributions, p_{in} and β. The p_{dm} for a state is upper bounded in Eq. (10.38). Each job's deadline miss probability is accounted for with the accumulation sequence resulting from the job's arrival. This is the case even with long relative deadlines, when actual deadlines are not missed until after the arrivals of subsequent jobs.

The iterative workload accumulation process connects all these different parts, and is presented in Section 10.5 with an example. An illustration of the process with references to relevant sections is provided in Fig. 10.3. The workload distribution bounds from Section 10.4.4 are used in all remaining

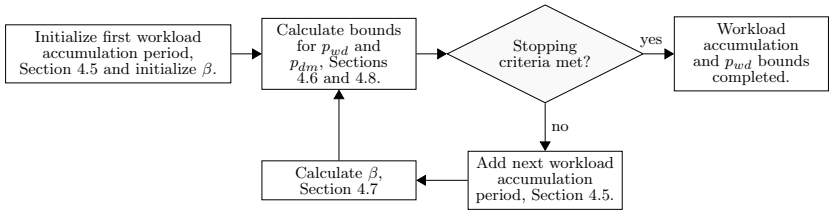


Fig. 10.3: The workload accumulation process.

sections and are not specifically referenced in the figure.

10.4.4 Bounding the Conditional Pending Workload Distribution Associated with a Workload Accumulation Sequence

Upper and lower bounds of pending workload distributions conditioned on the job arrival resulting in a given accumulation sequence since the last idle point are derived. To derive upper bounds on p_{dm} we require the upper bounds on the workload distributions. To derive upper and lower bounds on p_{in}, p_{co}, β and p_{wd} we require upper and lower bounds on the workload distributions. With an example from Fig. 10.2, we consider the pending workload distribution for jobs arriving in state 1, in the second task period from the last idle point, given that the first job after the idle point arrived in state 2, that is the path marked as gray, $h = (2, 1)$.

Definition 10.4.6. *The conditional pending workload distribution $\mathcal{V}_h$ for jobs resulting in a given accumulation sequence h has the probability density function $\mathbb{P}(v|\mathcal{H} = h)$.*

We will derive bounds for this conditional pending workload distribution that are independent of the order of the state visits in the accumulation sequence, and only dependent on the number of visits in each state. For this purpose we define the random variable $\tilde{\mathcal{H}}$ that takes S-dimensional vector values, where each element denotes the number of visits in the corresponding state since the last idle point. As an example, the dashed line in Fig. 10.2 showing $h = (2, 2, 1, 2, 2)$ and another accumulation sequence $h = (2, 2, 2, 1, 2)$

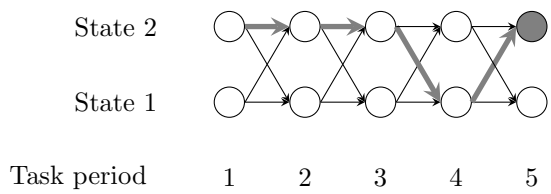


Fig. 10.4: Illustration of a workload accumulation sequence that contributes to the same workload accumulation vector as the dashed sequence in Fig. 10.2

illustrated in Fig. 10.4 contribute to the same accumulation vector. Both sequences result from jobs arriving 5 task periods after the latest idle point, and they have the same number of visits in each state, $\tilde{h} = [1, 4]$. Let $\tilde{h}[s]$ denote taking the s-th element of $\tilde{h}$, and $\tilde{h}_{+s}$ is the accumulation vector with elements: $\tilde{h}_{+s}[i] = \tilde{h}[i], i \neq s, \tilde{h}_{+s}[s] = \tilde{h}[s] + 1$. This simplifies the notation of the workload distribution of jobs arriving in state s with carry-in workload from $\tilde{h}$.

In a system with S states, the number of accumulation vectors of length N is $\binom{N+S-1}{N} = \frac{(N+S-1)!}{N!(S-1)!}$. If we take ordering into account, there are S^N accumulation sequences of length N. The number of accumulation vectors increases with the length N as $\mathcal{O}(N^{S-1})$ for a fixed number of states S.

We derive an upper bounding pending workload distribution $\mathcal{V}^{\uparrow}_{\tilde{h}} \geq \mathcal{V}_h$, recalling Definition 10.3.1.

We show that a *partial Gaussian distribution* (see Definition 10.3.2) is an upper bound to the conditional pending workload distribution. An illustration based on Example 10.4.1 is shown in Fig. 10.5. The dashed curve illustrates the exact convolution result of the workload of the gray accumulation sequence from Fig. 10.2. The carry-in workload is the partial Gaussian distribution of $\mathcal{N}^{tail}(2 - n \cdot Q, 1, 0)$, that is the normalized part of the computation time distribution in the second state that remains after the budget of $n \cdot Q = 2$ has been exhausted. The carry-in distribution is convolved with the computation time distribution of the first state, $\mathcal{N}(1, 0.25)$, resulting in the dashed curve. When we replace it with the partial Gaussian distribution shown in Fig. 10.5 as the black curve and line, the probabilities of lower workloads (the light gray area) are moved to higher workloads (the dark gray area), providing an upper bound.

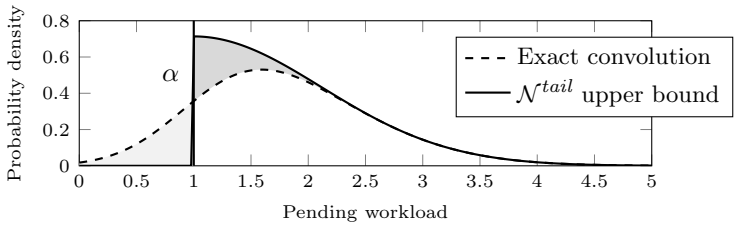


Fig. 10.5: Illustration of a convolution result with an upper bounding partial Gaussian distribution.

Theorem 10.4.1. *$\mathcal{N}^{tail}\left(\mu(\tilde{h}), \sigma^2(\tilde{h}), \alpha(\tilde{h}, s)\right)$ upper bounds the conditional pending workload distribution $\mathcal{V}_{\tilde{h}}$ associated with each state s and accumulation vector $\tilde{h}$.*

The proof is by induction. We state Lemma 10.4.2 for the base case and further Lemma 10.4.3 combined with Lemma 10.4.4 for the inductive step.

Lemma 10.4.2. *The partial Gaussian distribution $\mathcal{N}^{tail}\left(\mu_s, \sigma_s^2, 0\right) \geq \mathcal{V}_{\tilde{h}}$ in state s at a job arrival immediately after a point of workload depletion, with $\mathcal{V}_{\tilde{h}}$ being the conditional pending workload distribution.*

Proof. At the first job arrival after a point of workload depletion, the conditional pending workload distribution $\mathcal{V}_h$ is the execution time distribution of the entered state s. $\mathcal{N}^{tail}\left(\mu_s, \sigma_s^2, 0\right)$ excludes the negative workload values from $\mathcal{N}\left(\mu_s, \sigma_s^2\right)$. Normalization increases the probability of positive values. The probability density is moved from lower workload values to higher, providing an upper bound. □

In the following, we consider the case with non-zero carry-over workload when a job arrives in state s transitioning from state s_p. In s_p the accumulation vector is $\tilde{h}$, and $\mathcal{N}^{tail}(\mu(\tilde{h}), \sigma^2(\tilde{h}), \alpha(\tilde{h}, s_p))$ upper bounds the workload distribution. We show that the partial Gaussian distribution $\mathcal{N}^{tail}(\mu(\tilde{h}_{+s}), \sigma^2(\tilde{h}_{+s}), \alpha(\tilde{h}_{+s}, s))$ is an upper bound on the conditional pending workload distribution. In Eqs. (10.7) and (10.8) below we define $\mu(\tilde{h}_{+s})$ and $\sigma^2(\tilde{h}_{+s})$. To simplify the starting value $\alpha(\tilde{h}_{+s}, s)$ of the upper bound on

the pending workload distribution defined in Eq. (10.11), we define Eqs. (10.9) and (10.10). $\text{sf}^{-1}(q, \mu, \sigma^2)$ in Eq. (10.11) denotes the inverse survival function at quantile q of a Gaussian distribution with mean μ, and variance σ^2. The work value at which the survival function takes the value q. Eq. (10.10) defines $K(\tilde{h}, s_p)$, the normalization factor needed for the conditional probability calculation. A convolution Definition 10.3.3 of the execution time distribution C_s and an upper bound of the carry-over workload gives a bound on the pending workload distribution. The part extending past the task period of the upper bounding workload distribution in s_p with $\tilde{h}$ constitutes an upper bound of the carry-over workload. $K(\tilde{h}, s_p)^{-1}$ is the integral of this part used for normalization.

$$\mu(\tilde{h}_{+s}) = \mu_s + \sum_{i=1}^{S} \tilde{h}[i] \cdot (\mu_i - n \cdot Q) \tag{10.7}$$

$$\sigma^2(\tilde{h}_{+s}) = \sigma_s^2 + \sum_{i=1}^{S} \tilde{h}[i] \cdot \sigma_i^2 \tag{10.8}$$

$$\alpha_\Delta(\tilde{h}, s_p) = \max(0, \alpha(\tilde{h}, s_p) - n \cdot Q) \tag{10.9}$$

$$K(\tilde{h}, s_p) = \left[\Phi \left(\frac{\mu(\tilde{h}) - n \cdot Q - \alpha_\Delta(\tilde{h}, s_p)}{\sigma(\tilde{h})} \right) \right]^{-1} \tag{10.10}$$

$$\alpha(\tilde{h}_{+s}, s) = \begin{cases} 0 & \tilde{h} = \mathbf{0} \\ \text{sf}^{-1}(\frac{1}{K(\tilde{h}, s_p)}, \mu(\tilde{h}_{+s}), \sigma^2(\tilde{h}_{+s})) & \tilde{h} \neq \mathbf{0} \end{cases}. \tag{10.11}$$

Lemma 10.4.3. *When a job arrives in state s with non-zero carry-over workload from state s_p with accumulation vector $\tilde{h}$, and the previous task period upper bound on the workload distribution $\mathcal{V}^{\uparrow}$ as $\mathcal{N}^{tail}(\mu(\tilde{h}), \sigma^2(\tilde{h}), \alpha(\tilde{h}, s_p))$, the conditional pending workload distribution is upper bounded by $\mathcal{N}^{tail}(\mu(\tilde{h}_{+s}), \sigma^2(\tilde{h}_{+s}), \alpha(\tilde{h}_{+s}, s))$.*

Proof. The normalized workload tail beyond the task period time is the strictly positive carry-over workload distribution. We formally express this as $\mathcal{N}^{tail}\Big(\mu(\tilde{h}) - n \cdot Q\ ,\ \sigma^2(\tilde{h})\ ,\ \max(0, \alpha(\tilde{h}, s_p) - n \cdot Q)\Big)$.

$\mathcal{N}(\mu_s, \sigma_s^2)$ describes the execution time distribution in state s. By convolving Definition 10.3.3 the probability density functions of the execution time

and the upper bound on the positive carry-over workload, we derive an upper bound on the conditional workload distribution $\mathcal{V}^{\uparrow}_{\tilde{h}_{+s}}$ in state s with accumulation vector $\tilde{h}_{+s}$. This holds because execution times are independent random variables, and the dependence of the Markov model is restricted to the transition probabilities.

We introduce $\mu_R(z)$, σ_R^2, $\mu_{\Sigma\Delta}$ and σ_Σ^2 below to simplify the notation in the convolution expansion:

$$\mu_R(z) = \frac{(z-\mu_s)\cdot\sigma^2(\tilde{h}) + (\mu(\tilde{h}) - n\cdot Q)\cdot\sigma_s^2}{\sigma_s^2+\sigma^2(\tilde{h})} \tag{10.12}$$

$$\sigma_R^2 = \frac{\sigma_s^2\cdot\sigma^2(\tilde{h})}{\sigma_s^2+\sigma^2(\tilde{h})} \tag{10.13}$$

$$\mu_{\Sigma\Delta} = \mu_s + \mu(\tilde{h}) - n\cdot Q \tag{10.14}$$

$$\sigma_\Sigma^2 = \sigma_s^2 + \sigma^2(\tilde{h}). \tag{10.15}$$

We expand the convolution for $\mathcal{V}^{\uparrow}_{\tilde{h}_{+s}}$:

$$\begin{aligned}
&\int_{-\infty}^{\infty} f\left(z-x|\mu_s,\sigma_s^2\right)\cdot f^{tail}\left(x|\mu(\tilde{h})-n\cdot Q,\sigma^2(\tilde{h}),\alpha_\Delta\right)\mathrm{d}x \\
&= K(\tilde{h},s_p)\int_{\alpha_\Delta}^{\infty} f(z-x|\mu_s,\sigma_s^2)\cdot f(x|\mu(\tilde{h})-nQ,\sigma^2(\tilde{h}))\mathrm{d}x \\
&= K(\tilde{h},s_p)\cdot f\left(z|\mu_{\Sigma\Delta},\sigma_\Sigma^2\right)\cdot\int_{\alpha_\Delta}^{\infty} f\left(x|\mu_R(z),\sigma_R^2\right)\mathrm{d}x,
\end{aligned} \tag{10.16}$$

where we isolate the part of the expression independent of x in the last step. The integral in the last row of Eq. (10.16) is the survival function or 1-CDF at α_Δ of $\mathcal{N}(\mu_R(z),\sigma_R^2)$. The survival function is monotonically increasing with respect to z and goes to 0 as z goes to $-\infty$, and to 1 as z goes to ∞. This implies that there is a point $\alpha(\tilde{h}_{+s},s)$ where the area under the curve of the exact convolution of the pending workload distribution up to $\alpha(\tilde{h}_{+s},s)$ equals the area between the curves of the exact pending workload distribution and the partial Gaussian distribution, $\mathcal{N}^{tail}(\mu_{\Sigma\Delta},\sigma_\Sigma^2,\alpha(\tilde{h}_{+s},s))$ from $\alpha(\tilde{h}_{+s},s)$. This is illustrated in Fig. 10.5. Normalizing the partial Gaussian distribution with $K(\tilde{h},s_p)$

as in Eq. (10.17) means that we derive the lowest possible $\alpha(\tilde{h}_{+s}, s)$ that upper bounds the full convolution. As the integral in the last row of Eq. (10.16) goes to 1 as z goes to infinity, the tail of the upper bound approaches the tail of the full convolution asymptotically.

$$K(\tilde{h}, s_p) \cdot \int_{\alpha(\tilde{h}_{+s},s)}^{\infty} f\left(x|\mu_{\Sigma\Delta}, \sigma_{\Sigma}^2\right) \, \mathrm{d}x = 1. \tag{10.17}$$

The convolution result integrates to one, and so does the partial Gaussian distribution from Definition 10.3.2. The two regions described and illustrated in Fig. 10.5 have the same area. Replacing the exact convolution with the partial Gaussian is equivalent to moving probability weight from lower pending workload values to higher, leading to an overestimate. We have:

$$\mu(\tilde{h}_{+s}) = \mu_{\Sigma\Delta} \tag{10.18}$$

$$\sigma^2(\tilde{h}_{+s}) = \sigma_{\Sigma}^2 \tag{10.19}$$

$$\alpha(\tilde{h}_{+s}, s) = \mathrm{sf}^{\text{-}1}\left(\frac{1}{K(\tilde{h}, s_p)}, \mu_{\Sigma\Delta}, \sigma_{\Sigma}^2\right). \tag{10.20}$$

This concludes our proof. □

The values of α and K depend on the order in the accumulation sequence, as Eq. (10.9) depends on the previous state. Returning to Example 10.4.1, consider a job arriving in the third task period after an idle point. Two accumulation sequences determine the carry-in workload of visiting both state 1 and state 2 since the idle point; those are $h = (2, 1)$ and $h(1, 2)$. The first is the gray sequence in Fig. 10.2, and the upper bounding workload distribution is shown in Fig. 10.5. The tail extending past the period's budget is the carry-in to the next period, illustrated as the dashed curve in Fig. 10.6. Since $\alpha = 1 < n \cdot Q = 2$, $\alpha_{\Delta} = 0$ for the sequence $(2, 1)$. For the order $(1, 2)$ however, the resulting $\alpha \approx 3.236 > n \cdot Q = 2$. The upper bounding carry-in work will have $\alpha_{\Delta} \approx 1.236$ and is the solid curve illustrated in Fig. 10.6.

We state this formally in Lemma 10.4.4. We show that shifting the starting point α to a higher value while keeping the mean and variance unchanged gives an upper bounding distribution. This is illustrated in Fig. 10.7.

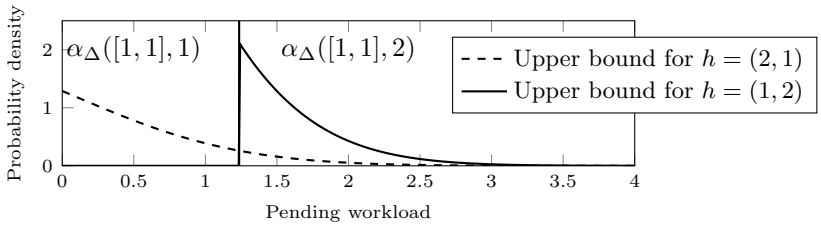


Fig. 10.6: Illustration of upper bounding partial Gaussian distributions for the carry-in workload of two accumulation sequences with the same vector.

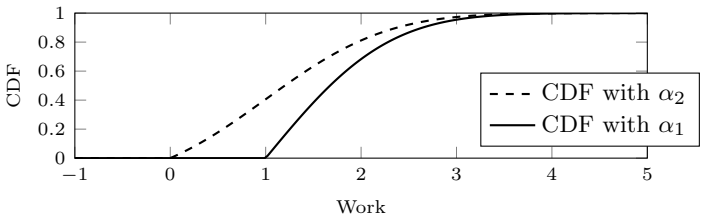


Fig. 10.7: CDFs of two partial Gaussian distributions as in Lemma 10.4.4. In this figure $\mu = 1$, $\sigma^2 = 1$, $\alpha_1 = 1$ and $\alpha_2 = 0$.

Lemma 10.4.4. *The partial Gaussian distribution* $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_1) \geq \mathcal{N}^{tail}(\mu, \sigma^2, \alpha_2)$ *if* $\alpha_1 \geq \alpha_2$.

Proof. The CDF is 0, $x < \alpha_2$ for both $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_1)$ and $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_2)$. The CDF of $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_2) > 0$ for $\alpha_2 \leq x \leq \alpha_1$, but the CDF of $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_1) = 0$ in this range. For $x > \alpha_1$, we have from Definition 10.3.2 that the PDFs of the two distributions only differ in the scaling factor. This means that the CDF of $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_1)$ is the CDF of $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_2)$ past α_1 shifted to start at 0 and scaled to go to 1 at infinity. Therefore the CDF of $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_1)$ is always below the CDF of $\mathcal{N}^{tail}(\mu, \sigma^2, \alpha_2)$

□

We remove the dependency on the state order by taking the maximum $\alpha(\tilde{h}, s_p), s_p \in \tilde{h}$ to determine $\alpha_\Delta(\tilde{h})$. This is illustrated by selecting the black

curve in Fig. 10.6 as carry-in from $\tilde{h} = [1, 1]$, and formalized as:

$$\alpha_{\Delta}(\tilde{h}) = \max(0, \max_{\forall s_p} \alpha(\tilde{h}, s_p) - n \cdot Q). \tag{10.21}$$

We use this instead of Eq. (10.9) in Eqs. (10.10) and (10.11). Let us proceed to the proof of Theorem 10.4.1, restated here for convenience:

Theorem 10.4.1. *$\mathcal{N}^{tail}\left(\mu(\tilde{h}), \sigma^2(\tilde{h}), \alpha(\tilde{h}, s)\right)$ upper bounds the conditional pending workload distribution $\mathcal{V}_{\tilde{h}}$ associated with each state s and accumulation vector $\tilde{h}$.*

Proof. We prove this by induction.
Base case: For the first job arrival after workload depletion, this follows by Lemma 10.4.2.
Inductive hypothesis: If we have such a workload distribution upper bound for all states and accumulation vectors in one task period, it also holds for a job that arrives with a carry-in workload from a previous period.
Inductive step: This follows from Lemma 10.4.3 and by taking the maximum α in Eq. (10.21) due to Lemma 10.4.4. □

Analogously, a Gaussian distribution is a lower bound of the pending workload distribution $\mathcal{V}^{\downarrow}_{\tilde{h}} \leq \mathcal{V}_h$. This is illustrated in Fig. 10.8 for the accumulation sequence example drawn in gray in Fig. 10.2. From Eq. (10.16), we see that $K(\tilde{h}, s_p) > 1$, implying a heavier tail on the convolution result compared to the Gaussian distribution. The area under the Gaussian PDF curve with mean $\mu_{\Sigma\Delta}$ and variance σ^2_{Σ} is one, and so is the area under the result of the convolution. Replacing the workload distribution with the Gaussian implies moving probability weight from higher workload values to lower, thus providing a lower bound.

10.4.5 Bounds on the Joint Probability of a Job Arriving in a State with an Accumulation Vector

A job arriving in state s N task periods after the last workload depletion can result in one or more accumulation vectors, $\tilde{h}$, of length N. We refer to this set

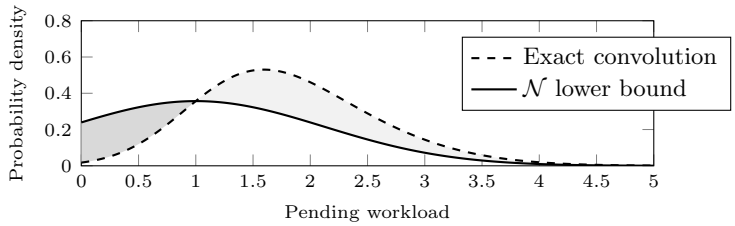


Fig. 10.8: An illustration of a convolution result and the Gaussian distribution that forms a lower bound.

of accumulation vectors as being in state s at task period N. Each accumulation vector in a state is associated with the joint probability of randomly selecting a job that arrives in state s and results in the accumulation vector $\tilde{h}$. $p^{\downarrow}_{in}(s, \tilde{h})$ denotes a lower bound on this joint probability and $p^{\uparrow}_{in}(s, \tilde{h})$ an upper bound. Each accumulation vector in a state is also associated with a probability of the workload contributing to carry-over into the next period. $p^{\downarrow}_{co}(s, \tilde{h})$ denotes a lower bound on this probability and $p^{\uparrow}_{co}(s, \tilde{h})$ an upper bound.

For jobs arriving at a point of workload depletion with no carry-in workload, each state is associated with a single accumulation vector containing zeros except for the current state, which is set to 1. The probability of a job arriving in a certain state s at a point of workload depletion depends on

- the stationary probabilities $\xi(s_p)$ of all states s_p,
- the workload depletion probabilities $p_{wd}(s_p)$ of all states s_p,
- the state transition probabilities $\xi_{s_p,s}$ from all states s_p into s.

The stationary probabilities and the transition matrix are known from the execution time model described in Section 10.4.1. In Section 10.4.6, we will describe how to derive the workload depletion probabilities of all states. Let us assume that we have lower and upper bounds on the workload depletion probabilities, $p^{\downarrow}_{wd}(s)$ and $p^{\uparrow}_{wd}(s)$. Then, lower and upper bounds on the probability of randomly selecting a job arriving in each state s at a point of workload depletion

are given as:

$$p^{\downarrow}_{in}(s,\tilde{h}) = \sum_{s_p=1}^{S} \xi(s_p) \cdot p^{\downarrow}_{wd}(s_p) \cdot m_{s_p,s} \tag{10.22}$$

$$p^{\uparrow}_{in}(s,\tilde{h}) = \sum_{s_p=1}^{S} \xi(s_p) \cdot p^{\uparrow}_{wd}(s_p) \cdot m_{s_p,s}. \tag{10.23}$$

There is only one accumulation vector in each state for jobs arriving at an idle point, and there is no dependency on $\tilde{h}$. We introduce it in the expression to have the common notation $p_{in}(s,\tilde{h})$ for all accumulation periods.

Relating this to Example 10.4.1, the lower bound on the probability of a job arriving in state 2 after an idle point is $p^{\downarrow}_{in}(2,[0,1]) = \xi(1) \cdot m_{1,2} \cdot p^{\downarrow}_{wd}(1) + \xi(2) \cdot m_{2,2} \cdot p^{\downarrow}_{wd}(2) = 0.875 \cdot 0.1 \cdot p^{\downarrow}_{wd}(1) + 0.125 \cdot 0.3 \cdot p^{\downarrow}_{wd}(2)$, a linear combination of the lower bounds on workload depletion probability for the states. The upper bound is the same linear combination of the upper bounds on workload depletion.

We further consider jobs arriving with a carry-in workload. Step by step, we add jobs that arrive one more task period after the last idle point, resulting in accumulation vectors containing one more state. We copy each accumulation vector from the states in the previous task period for these accumulation periods and increment the current state element by 1. Such an accumulation vector copied from $\tilde{h}$ with a job that arrives in state s is denoted $\tilde{h}_{+s}$. When $\tilde{h}$ exists in different states in the previous accumulation period, they all lead to $\tilde{h}_{+s}$ in s. The joint probability of randomly selecting a job arriving in s and resulting in $\tilde{h}_{+s}$ depends on the probabilities $p_{co}(s_p,\tilde{h})$ of a randomly selected job arriving with unfinished workload from $\tilde{h}$ in each state s_p, and transition probabilities $m_{s_p,s}$.

The probability that a job arrives with a carry-in workload from $s_p,\tilde{h}$ is the probability of being in s_p with this $\tilde{h}$ multiplied by the probability that the conditional pending workload of $s_p,\tilde{h}$ exceeds the available processor time in a task period. Let random variables $X \sim \mathcal{V}^{\downarrow}_{\tilde{h}}$ and $Y \sim \mathcal{V}^{\uparrow}_{\tilde{h}}$. Then we have lower $p^{\downarrow}_{co}(s,\tilde{h})$ and upper $p^{\uparrow}_{co}(s,\tilde{h})$ bounds on the probability of a job arriving with the carry-in workload from $\tilde{h}$ and where the previous task period state was s as:

$p^{\downarrow}_{co}(s,\tilde{h})$ and $p^{\uparrow}_{co}(s,\tilde{h})$, further calculated as:

$$p^{\downarrow}_{co}(s,\tilde{h}) = p^{\downarrow}_{in}(s,\tilde{h}) \cdot \mathbb{P}(X > n \cdot Q) \tag{10.24}$$

$$p^{\uparrow}_{co}(s,\tilde{h}) = p^{\uparrow}_{in}(s,\tilde{h}) \cdot \mathbb{P}(Y > n \cdot Q) \tag{10.25}$$

with $\mathcal{V}^{\downarrow}_{\tilde{h}}$ given as $\mathcal{N}(\mu(\tilde{h}),\sigma^2(\tilde{h}))$, and $\mathcal{V}^{\uparrow}_{\tilde{h}}$ as $\mathcal{N}^{tail}(\mu(\tilde{h}),\sigma^2(\tilde{h}),\alpha(\tilde{h}))$.

In Example 10.4.1 we find the lower bound of the probability of workload carry-over from state 2 and $\tilde{h} = [0,1]$ as $p^{\downarrow}_{co}(2,[0,1]) = p^{\downarrow}_{in}(2,[0,1]) \cdot \mathbb{P}(\mathcal{N}(2,1) > 2) = 0.5 \cdot p^{\downarrow}_{in}(2,[0,1])$. The upper bound is $p^{\uparrow}_{co}(2,[0,1]) = p^{\uparrow}_{in}(2,[0,1]) \cdot \mathbb{P}(\mathcal{N}^{tail}(2,1,0) > 2) \approx 0.51 \cdot p^{\uparrow}_{in}(2,[0,1])$.

The lower $p^{\downarrow}_{in}(s,\tilde{h}_{+s})$ and upper $p^{\uparrow}_{in}(s,\tilde{h}_{+s})$ bounds on the joint probability a job arriving in s resulting in $\tilde{h}_{+s}$ are:

$$p^{\downarrow}_{in}(s,\tilde{h}_{+s}) = \sum_{s_p=1}^{S} p^{\downarrow}_{co}(s_p,\tilde{h}) \cdot m_{s_p,s} \tag{10.26}$$

$$p^{\uparrow}_{in}(s,\tilde{h}_{+s}) = \sum_{s_p=1}^{S} p^{\uparrow}_{co}(s_p,\tilde{h}) \cdot m_{s_p,s}. \tag{10.27}$$

Returning to Example 10.4.1 and the gray accumulation sequence in Fig. 10.2, we have the probability of a job arriving in state 1 with carry-in from one task period in state 2. The lower bound on this probability is $p^{\downarrow}_{in}(1,[1,1]) = p^{\downarrow}_{co}(2,[0,1]) \cdot m_{2,1} = 0.7 \cdot p^{\downarrow}_{co}(2,[0,1])$. In this case, the sum has only one term since only $\tilde{h} = [0,1]$ in the first period can lead to $\tilde{h} = [1,1]$ and $s = 1$ in the second period. The upper bound is derived in the same manner as $p^{\uparrow}_{in}(1,[1,1]) = 0.7 \cdot p^{\uparrow}_{co}(2,[0,1])$. The derivations from Eqs. (10.22), (10.24) and (10.26) can be combined, giving $p^{\downarrow}_{in}(1,[1,1]) \approx 0.0306 p^{\downarrow}_{wd}(1) + 0.0131 p^{\downarrow}_{wd}(2)$. Combining Eqs. (10.23), (10.25) and (10.27) gives $p^{\uparrow}_{in}(2,[1,1]) \approx 0.0313 p^{\uparrow}_{wd}(1) + 0.0134 p^{\uparrow}_{wd}(2)$.

10.4.6 Bounds on the Probability of Workload Depletion

The probability of having no work remaining at the end of the task period p_{wd} for each state are used in Eqs. (10.22) and (10.23) to derive p_{in} bounds for jobs

arriving at idle points. These state-wise workload depletion probabilities p_{wd} are propagated to all p_{in} in the workload accumulation process via Eqs. (10.24) to (10.27).

We do not know the true value of the probability of workload depletion p^*_{wd}. This section outlines how to derive bounds for p_{wd} by observing bounds of state-wise sums on p_{in}.

Let $\tilde{h} \in (s, i)$ denote the set of accumulation vectors in state s at task period i from the last idle point. We now define $p^{\downarrow\Sigma}_{in}(s, N, p_{wd})$, the sum of the lower bounds on p_{in} associated with all accounted accumulation vectors in s up until task period N from the last idle point. In other words, this is a lower bound on the joint probability of a randomly selected job arriving in s and at most N from the last idle point.

$$p^{\downarrow\Sigma}_{in}(s, N, p_{wd}) = \sum_{i=1}^{N} \sum_{\tilde{h}\in(s,i)} p^{\downarrow}_{in}(s, \tilde{h}) \tag{10.28}$$

Observation 10.4.4. *Assume the exact p^*_{wd} is known and used as $p^{\downarrow}_{wd}$ in Eq.* (10.22)*. Then $p^{\downarrow\Sigma}_{in}(s, N, p_{wd}) \leq \xi(s), \forall s, \forall N$.*

We denote the error in $p^{\downarrow\Sigma}_{in}$ resulting from using the Gaussian $\mathcal{V}^{\downarrow}$ lower workload distribution bounds instead of the true workload distributions as $e(p^{\downarrow\Sigma}_{in})$.

We introduce $\beta(s)_N$ as the joint probability of a job arriving in s more than N task periods after the last idle point.

$$\beta(s)_N = \sum_{i=N+1}^{\infty} \sum_{\tilde{h}\in(s,i)} p_{in}(s, \tilde{h}) \tag{10.29}$$

Observation 10.4.4 is illustrated in Fig. 10.9, where the valid region of $p^{\downarrow\Sigma}_{in}$ is displayed assuming p^*_{wd} is input in Eq. (10.22).

We define an upper bound on the joint probability of a randomly selected job arriving in s and at most N from the last idle point as $p^{\uparrow\Sigma}_{in}(s, N, p_{wd})$ in Eq. (10.30).

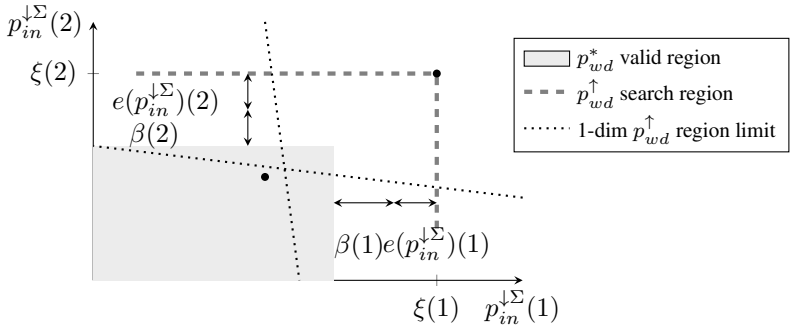


Fig. 10.9: An illustration of the possible valid region of $p_{in}^{\downarrow\Sigma}$ for two states, if the true probabilities of workload depletion would be used as $p_{wd}^{\downarrow}$ in Eq. (10.22).

$$p_{in}^{\uparrow\Sigma}(s, N, p_{wd}) = \sum_{i=1}^{N} \sum_{\tilde{h}\in(s,i)} p_{in}^{\uparrow}(s, \tilde{h}) \tag{10.30}$$

Observation 10.4.5. *Assume the true probability of workload depletion p_{wd}^{*} is known. Using this value in Eq.* (10.23)*, we have $p_{in}^{\uparrow\Sigma}(s, N, p_{wd}) \geq \xi(s) - \beta(N)$.*

Let $e(p_{in}^{\uparrow\Sigma})$ denote the error introduced by replacing the true workload distribution with the upper bounding partial Gaussian distribution. The valid region of $p_{in}^{\uparrow\Sigma}$ given from observation 10.4.5 is displayed in Fig. 10.10.

Observations 10.4.4 and 10.4.5 imply that the true probability of workload depletion must lead to $p_{in}^{\downarrow\Sigma}$ in the region marked in Fig. 10.9 and $p_{in}^{\uparrow\Sigma}$ in the region marked in Fig. 10.10. The state-wise maxima of p_{wd} leading to any point along the lines illustrated as the upper and right lines in Fig. 10.9 upper bounds p_{wd}^{*}.

Theorem 10.4.5. *An upper bound $p_{wd}^{\uparrow}$ on the probability of workload depletion is derived by taking the state-wise maxima of p_{wd} leading to $p_{in}^{\downarrow\Sigma}(s) \leq \xi(s), \forall s$, and where there is inequality in at most one s.*

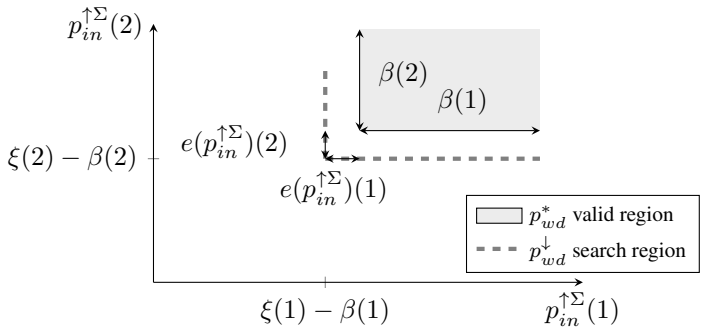


Fig. 10.10: An illustration of the possible valid region of $p_{in}^{\uparrow\Sigma}$ for two states, if the true probabilities of workload depletion would be used as $p_{wd}^{\uparrow}$ in Eq. (10.23).

Proof. From Eqs. (10.22), (10.24) and (10.26) it follows that $p_{in}^{\downarrow}(s, \tilde{h})$ is a linear combination of $p_{wd}(s)$. As is clear from Eq. (10.28), $p_{in}^{\downarrow\Sigma}(s)$ is also a linear combination of $p_{wd}(s)$, and it holds for some positive $A_{i,s}$ that:

$$p_{in}^{\downarrow\Sigma}(s, p_{wd}) = \sum_{i=1}^{S} A_{i,s} \cdot p_{wd}(i) \tag{10.31}$$

Starting from the true workload depletion probability p_{wd}^{*} we increase an arbitrary state dimension j of $p_{wd}(j)$ by an amount $\delta_{s,j}$ until $p_{in}^{\downarrow\Sigma}(s, p_{wd})$ reaches a hyperplane defined by ξ:

$$p_{in}^{\downarrow\Sigma}(s, p_{wd}) = A_{j,s}(p_{wd}^{*}(j) + \delta_{s,j}) + \sum_{i=1, i\neq j}^{S} A_{i,s} p_{wd}^{*}(i) = \xi(s)$$

At the first hyperplane we encounter $\min(\delta_{s,j}) \forall s$, which gives $p_{in}^{\downarrow\Sigma}(i) \leq \xi(i), \forall i \neq s$.

The true p_{wd}^{*} results in $p_{in}^{\downarrow\Sigma}(s) \leq \xi(s)$. Therefore, all p_{wd} resulting in the point with equality for all s upper bounds p_{wd}^{*} in at least one state dimension due to the linear combination. Assume that p_{wd} resulting in this point does not

upper bound p^*_{wd} for state dimension i, $p_{wd}(i) < p^*_{wd}(i)$. In this case, an upper bound of $p^*_{wd}(i)$ results in a point on one of the hyperplanes. The hyperplane separating the region resulting from upper bounds in this dimension from the region resulting from underestimates in this dimension crosses at least one of the hyperplanes described by $p^{\downarrow\Sigma}_{in}(s) \leq \xi(s), \forall s$, with inequality in at most one s. Illustrating in Fig. 10.9 the result from the upper bound in this dimension as the black dot, two possible hyperplanes that separate the regions are displayed with dotted lines. This concludes our proof. □

Analogously, we derive a lower bound on the workload depletion probability p_{wd}. The state-wise minima of p_{wd} resulting in $p^{\uparrow\Sigma}_{in}$ on the lower and left lines illustrated in Fig. 10.10 lower bound p^*_{wd}.

The endpoints are adjusted if the p_{wd} for a state is lower than 0 or higher than 1. As $p^{\downarrow\Sigma}_{in}(s)$ and $p^{\uparrow\Sigma}_{in}(s)$ are linear combinations of $p_{wd}(s)$ we only need to consider the endpoints.

Relating to Example 10.4.1, we have seen that the probability of a job arrival in state 2 after an idle point at least $p^{\downarrow}_{in}(2, [0, 1]) = 0.0875 \cdot p^{\downarrow}_{wd}(1) + 0.0375 \cdot p^{\downarrow}_{wd}(2)$. The same derivation for a job arrival in state 1 after an idle point gives $p^{\downarrow}_{in}(1, [1, 0]) = 0.7875 \cdot p^{\downarrow}_{wd}(1) + 0.0875 \cdot p^{\downarrow}_{wd}(2)$. From simulation we have the probability of jobs arriving with $\tilde{h}$ longer than 1 as $\beta(1)_1 \approx 0.093$ for state 1 and $\beta(2)_1 \approx 0.026$ for state 2. We solve the linear equation systems below for $(i, j) = (0, 0), (1, 0)$ and $(0, 1)$ to get candidates for $p^{\uparrow}_{wd}$.

$$0.7875 \cdot p_{wd}(1) + 0.0875 \cdot p_{wd}(2) = 0.875 - i \cdot 0.093 \tag{10.32}$$

$$0.0875 \cdot p_{wd}(1) + 0.0375 \cdot p_{wd}(2) = 0.125 - j \cdot 0.026 \tag{10.33}$$

In this case, with only the jobs arriving at idle points, the equation system for $(i, j) = (0, 0)$, the upper right corner in Fig. 10.9, gives the solution $p^{\uparrow}_{wd}(1) = p^{\uparrow}_{wd}(2) = 1$ that is the highest possible bound. For the lower bound of p_{wd}, we have the same linear equation system in the special case when we only consider the accumulation vectors after an idle point. This is because Eqs. (10.22) and (10.23) only differ in the workload depletion probability bounds. We now solve the system for $(i, j) = (1, 1), (1, 0)$ and $(0, 1)$. For $(i, j) = (1, 1)$, the lower left corner in Fig. 10.10, we get the candidates

$p_{wd}(1) \approx 0.94$ and $p_{wd}(2) \approx 0.44$. For $(i,j) = (1,0)$ we get $p_{wd}(1) \approx 0.84$ and $p_{wd}(2) \approx 1.4$. This point is invalid since $p_{wd}(2) > 1$. We find the $j, 0 < j < 1$ where $(i,j) = (1,j)$ gives $p_{wd}(2) = 1$, and at this point we have $p_{wd}(1) \approx 0.88$. For $(i,j) = (0,1)$ we get $p_{wd}(1) \approx 1.1$ and $p_{wd}(2) \approx 0.064$. This point is also invalid, and we search along the line $i, 0 < i < 1, j = 1$ for the point where $p_{wd}(1) = 1$. We have $p_{wd}(2) \approx 0.31$. We can now assign the state-wise minima for the lower bound: $p^{\downarrow}_{wd}(1) \approx 0.88$ and $p^{\downarrow}_{wd}(2) \approx 0.31$.

10.4.7 Bounds on the Probability of Longer Workload Accumulation

In Equation (10.29), we defined $\beta(s)_N$ as the joint probability of a job arriving in s with at least N elapsed since the last idle point. The values of $\beta(s)_N$ were used in obtaining the bounds of workload depletion for the different states. In this section we outline how to find bounds for $\beta(s)_N$, given safe bounds $\beta(s)_{N-1}$. As all p_{in} are non-negative, $\beta(s)_N$ decreases monotonically with N. For each period, $\beta(s)_N$ is at most $\beta(s)_{N-1}$ minus the lower bound on the probability of a job arriving in s N task periods after an idle point, i.e.

$$\beta(s)_N \leq \beta(s)_{N-1} - \sum_{\tilde{h} \in (s,N)} p^{\downarrow}_{in}(s, \tilde{h}) = \beta(s)^{\uparrow a}_N \tag{10.34}$$

Further, $\beta(s)_N$ is at most the stationary probability $\xi(s)$ minus the lower bound on the probability of a job arriving within N task periods after an idle point, i.e.

$$\beta(s)_N \leq \xi(s) - \sum_{i=1}^{N} \sum_{\tilde{h} \in (s,i)} p^{\downarrow}_{in}(s, \tilde{h}) = \beta(s)^{\uparrow b}_N \tag{10.35}$$

Safe bounds $\beta(s)_N$ are obtained by taking the minimum of right-hand sides of Inequalities 10.34, and 10.35.

$$\beta(s)^{\uparrow}_N = \min(\beta(s)^{\uparrow a}_N, \beta(s)^{\uparrow b}_N) \tag{10.36}$$

We return to Example 10.4.1 and consider the probability of jobs arriving in state 1 with accumulation vectors past 2 task periods, that is $\beta(1)_2$. We have

$\beta(1)_1 \approx 0.093$ from simulation, the lower bound on the probability of a job arriving in state 1 with accumulation vector $[1,1]$ as $p^{\downarrow}_{in}(1,[1,1]) \approx 0.031 \cdot p^{\downarrow}_{wd}(1) + 0.013 \cdot p^{\downarrow}_{wd}(2)$ and with accumulation vector$[2,0]$ $p^{\downarrow}_{in}(1,[2,0]) \approx 0.016 \cdot p^{\downarrow}_{wd}(1) + 0.0018 \cdot p^{\downarrow}_{wd}(2)$. Entering these values in Eq. (10.34) we get $\beta(1)_2 \lessapprox 0.047$. Using Eq. (10.35) gives $\beta(1)_2 \lessapprox 0.11$, so we use $\beta(1)^{\uparrow}_2 \approx 0.047$ in the search for bounds on p_{wd} in the next accumulation period.

10.4.8 Upper Bounding the Deadline Miss Probability

Finally, we derive an upper bound on a randomly selected job's expected deadline miss probability as defined in Eq. (10.5). We derive an upper bound $p^{\uparrow}_{dm}(s,\tilde{h})$ on the deadline miss probability $p_{dm}(s,\tilde{h})$ of a job arriving in state s with the job arrival resulting in the accumulation vector $\tilde{h}$. This bounds the deadline miss probability of all jobs, resulting in accumulation sequences h corresponding to $\tilde{h}$ where the sequence ends in s. The random variable $Y \sim \mathcal{V}^{\uparrow}_{\tilde{h}}$ upper bounds the pending work distribution of these jobs. $p^{\uparrow}_{dm}(s,\tilde{h})$ is the probability that this work exceeds the available computation time for the job until the deadline $k \cdot Q$, i.e.:

$$p^{\uparrow}_{dm}(s,\tilde{h}) = \mathbb{P}(Y > k \cdot Q) \tag{10.37}$$

The distribution $\mathcal{V}^{\uparrow}_{\tilde{h}}$ is the upper bounding distribution $\mathcal{N}^{tail}(\mu(\tilde{h}), \sigma^2(\tilde{h}), \alpha(\tilde{h},s))$, as shown in Theorem 10.4.1.

The probability of randomly selecting a job arriving in state s with workload accumulation captured by $\tilde{h}$ is the joint probability $p_{in}(s,\tilde{h})$. The upper bound of this probability, $p^{\uparrow}_{in}(s,\tilde{h})$ was derived in Section 10.4.5. We derive a bound of the expected deadline miss probability conditioned on being in a state s by considering all job arrivals in s within N task periods from the last idle point, that is with $\tilde{h}$ of length up to N. The deadline miss probability of jobs arriving more than N task periods from the last idle point is upper bounded by 1. The probability of randomly selecting a job arriving more than N task periods from the last idle point is upper bounded by $\beta^{\uparrow}(s)_n Periods$. The probability of randomly selecting a job arriving in s is the stationary probability $\xi(s)$. We

upper bound the expected deadline miss probability in s by:

$$p^{\uparrow}_{dm}(s) = \frac{\beta(s)^{\uparrow}_{N}}{\xi(s)} + \frac{\sum_{i=1}^{N}\sum_{\tilde{h}\in(s,i)} p^{\uparrow}_{in}(s,\tilde{h})p^{\uparrow}_{dm}(s,\tilde{h})}{\xi(s)}. \tag{10.38}$$

In our example, we derive for state 1 the first term $\frac{\beta(1)^{\uparrow}_{2}}{\xi(1)} \approx \frac{0.047}{0.875} \approx 0.054$ and the second term $\frac{\sum_{\tilde{h}\in([1,0],[1,1],[2,0])} p^{\uparrow}_{in}(1,h)\cdot p^{\uparrow}_{dm}(1,h)}{\xi(1)} \approx \frac{0.875\cdot 10^{-9}+0.018\cdot 3.4\cdot 10^{-7}+0.045\cdot 0.0073}{0.875} \approx 3.7\cdot 10^{-4}$.

Theorem 10.4.6. *The expected deadline miss probability DMP of a randomly selected job is upper-bounded by $p^{\uparrow}_{dm}$, i.e., DMP $\leq p^{\uparrow}_{dm}$, where*

$$p^{\uparrow}_{dm} = \sum_{\forall s}\Big(\beta(s)^{\uparrow}_{N} + \sum_{i=1}^{N}\sum_{\tilde{h}\in(s,i)} p^{\uparrow}_{in}(s,\tilde{h})p^{\uparrow}_{dm}(s,\tilde{h})\Big). \tag{10.39}$$

Proof. The deadline miss probability Eq. (10.37) is an upper bound on the deadline miss probability of a job arriving in s and resulting in $\tilde{h}$, because $\mathcal{N}^{tail}(\mu(\tilde{h}),\sigma^2(\tilde{h}),\alpha(\tilde{h},s))$ upper bound on the workload distribution as shown in per Theorem 10.4.1.

The expected deadline miss probability of a randomly selected job arriving in s is upper bounded by $p^{\uparrow}_{dm}(s)$ as in Eq. (10.38). For jobs arriving in s within N since the last idle point, Eq. (10.37) upper bounds $p_{dm}(s,\tilde{h})$, and $p_{in}(s,\tilde{h})$ is an upper bound on the probability of randomly selecting a job arriving in state s and resulting in $\tilde{h}$. The probability of randomly selecting a job arriving in s more than N from the last idle point is upper bounded by $\beta(s)^{\uparrow}_{N}$, and 1 upper bounds $p_{dm}(s,\tilde{h})$ for these jobs. We divide by $\xi(s)$ as per the definition of conditional probability.

We apply the law of total probability on Eq. (10.38) over all the states s to obtain Eq. (10.39). □

In our example we have $\beta(1)^{\uparrow}_{2} \approx 0.047$ and $\beta(2)^{\uparrow}_{2} \approx 0.011$, resulting in the first term of Eq. (10.39) as 0.058. In the second term we have for state 2 $\sum_{\tilde{h}\in([0,1],[1,1],[0,2])} p^{\uparrow}_{in}(2,h)\cdot p^{\uparrow}_{dm}(2,h) \approx 0.125\cdot 0.024 + 0.045\cdot 0.16 + 0.019\cdot 0.16$. Summing with the terms of state one, the resulting sum is approximately

0.0066. With only two accumulation periods accounted for, the largest part of the bound stems from the first term, where the deadline miss probability is set to 1 for longer accumulation vectors.

10.5 Iterative Workload Accumulation

As illustrated in Fig. 10.3, the steps described in Section 10.4 are applied iteratively, successively including jobs arriving with a longer time from the last idle point in the analysis. The process ends when one of the following conditions is met:

1. For each state both of the following hold:
   (a) The upper bound on the probability of workload depletion has stopped decreasing and started increasing.
   (b) The lower bound on the probability of workload depletion has stopped increasing and started decreasing.
2. The process has reached a maximum number of accumulated periods.

If the bounds on the workload depletion probability converge for each state, or if the region within the bounds starts to grow, the first condition is met. Instead of performing the convolution in each accumulation period, the upper and lower bounds on the workload distribution are used, introducing an error. The white space between the valid region and the lines to use in searching for bounds in Figs. 10.9 and 10.10 illustrate these errors. When these errors increase, the distance between the search region for our bounds and the region resulting from the true workload depletion probabilities grows. For the upper bounds, illustrated in Fig. 10.9, the distance between the search region and the valid region may still decrease if the increase due to this error is compensated by a decrease in β. In the case of the lower bounds, illustrated in Fig. 10.10, a larger error leads to a smaller value for the lower bound. The lower bounds are used in the calculations of $\beta^{\uparrow}$ in the next accumulation period, Eq. (10.36). Smaller lower bounds lead to a larger $\beta^{\uparrow}$. This further increasing the distance between the valid region and the bound search region, as $\beta^{\uparrow}$ is used to determine the search region. This may cause $p_{wd}^{\downarrow}$ and $\beta^{\uparrow}$ to diverge.

It may be the case that the workload depletion probability bounds diverge from the beginning. This may be caused by insufficient computational resources allocated to the task or too large errors introduced in the bound calculations. It may also be the case that the workload depletion probability bounds converge slowly or converge for one or more states while they diverge for others. In these cases, the process stops when the second condition is fulfilled.

We apply the iterative process to the following example:

Example 10.5.1.

$$S = 2, \qquad \boldsymbol{M} = \begin{pmatrix} 0.9 & 0.1 \\ 0.7 & 0.3 \end{pmatrix}, \qquad \mathcal{C} = \{\mathcal{N}(20, 9), \mathcal{N}(40, 16)\}.$$

The stationary probability for state 1 *is* 0.875*, and for state* 2 *it is* 0.125*. The transition matrix and stationary probabilities are identical to Example 10.4.1. In the CBS of this example, there are* $n = 4$ *server periods in one task period, and the task is guaranteed* $Q = 8$ *time units of computation time in each server period. The deadline is defined by* $k = 8$ *server periods.*

State 1 in Example 10.5.1 could imply normal operation, and state 2 an exceptional mode. While in normal operation the task remains there with probability 0.9, but when the task is in the exceptional mode, there is a probability of 0.3 that it remains there. The initial values of probability of a randomly selected job arriving with carry-in workload from at least one task period, β_1 are obtained from simulation. Execution times are generated from the Markov Model and fed into a CBS simulator with the specified server reservation and period ratio. This results in $\beta_1 = (0.1278, 0.0442)$ for states 1 and 2, respectively. Figure 10.11 illustrated the evolution of β_N during the workload accumulation process compared to probabilities of jobs arriving in states 1 and 2 at least N from the last idle point resulting from simulation.

The bound regions for the probabilities of workload depletion of the two states along the accumulation process are shown in Fig. 10.12. Estimates of the probabilities of workload depletion obtained from simulation are also displayed. The workload accumulation continues until the maximum number of task periods, set to 20 for this example.

The bounds on the deadline miss probabilities for the two states along the accumulation process are shown in Fig. 10.13. The second terms of Eq. (10.38),

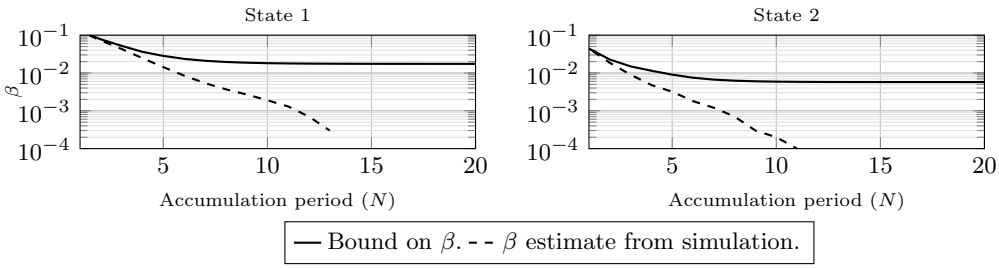


Fig. 10.11: Bounds on β for the two states as solid lines, along with probability estimates of longer accumulation histories obtained from simulation as dashed lines. (Log scale.)

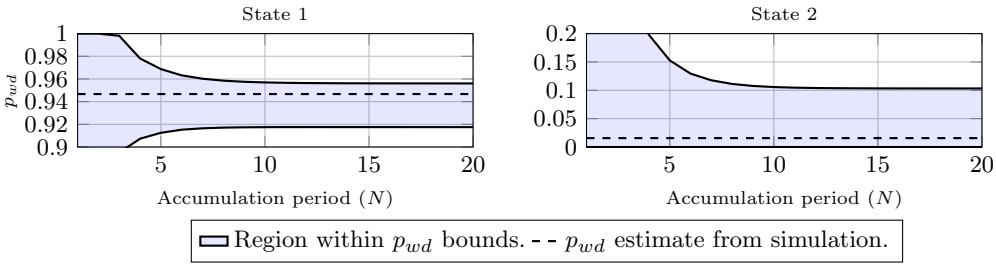


Fig. 10.12: The region between the upper and lower bounds on the per-state probability of workload depletion in the example, along with the estimates obtained from simulation as a dashed line.

the parts of the bounds resulting from the weighted sum of the accumulation vectors we have accounted for, are shown as dotted. In this example, the second terms approach the p_{dm} from simulation. The major part of the introduced pessimism originates in β, the first terms of Eq. (10.38). Estimates of the deadline miss probabilities obtained from simulation are also displayed in Fig. 10.13.

10.5.1 Time Complexity of the Iterative Process

In Section 10.4.4, we have seen that the number of accumulation vectors with length N in a S-state model grows as $\mathcal{O}(N^{S-1})$. In the iterative procedure,

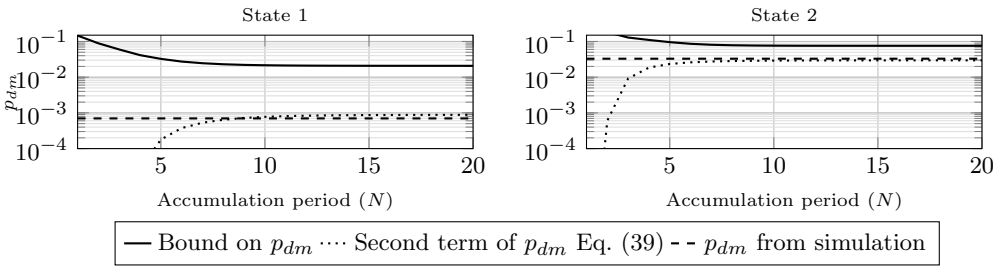


Fig. 10.13: The bounds on the deadline miss probabilities during the workload accumulation process of the example, along with results from simulation. (Log scale.)

all accumulation vectors up until length N have been considered at iteration step N, so the time complexity of the entire iterative process is $\mathcal{O}(N^S)$. In the current implementation, all vectors up until length N are considered in the iteration step N, giving a time complexity of $\mathcal{O}(N^{S+1})$. By storing intermediate results from previous iterations, this can be reduced to $\mathcal{O}(N^S)$. The number of states is application dependent. In the robotic vision task of Frías *et al.* [25], 4 discrete-emission states are identified, and in the control task in our evaluation 6 discrete-emission or 8 Gaussian-emission states are found. In the video decompression task evaluated in Friebe *et al.* [29] almost 50 Gaussian-emission states are identified.

10.6 Reducing the Number of States by Merging

As the time complexity of the iterative process up until the accumulation period N with a S-state model is $\mathcal{O}(N^S)$, it is clear that if the number of states can be reduced, this would have a great effect on the bound computation time. This section outlines how to reduce the number of states by merging while ensuring a safe bound on the deadline miss probability.

10.6.1 Modified Markov Chain Execution Times

In this section we define a modified execution time model, where an upper bound on the execution time distribution is defined by $\langle \mathbb{S}, M, \mathbb{C} \rangle$. As in the model defined in Section 10.4.1, $\mathbb{S} = \{1, 2, \ldots, S\}$ is the set of S states, $S \in \mathbb{N}$, and M is the $S \times S$ state transition matrix. $\mathbb{C} = \{C_1, C_2, \ldots, C_S\}$ is the set of upper bounding execution time distributions, or *emission distributions*, related to the respective state. These are modeled as partial Gaussian distributions with mean μ_s and variance σ_s^2 of the Gaussian distribution, and α_s as the starting point of the distribution, i.e. $C_s \sim \mathcal{N}^{tail}(\mu_s, \sigma_s^2, \alpha_s)$. Setting $\alpha_s = -\infty, \forall s$, gives the model as defined in Section 10.4.1.

10.6.2 Merging Distributions

Definition 10.6.1. *We define a merged partial Gaussian distribution* $\mathcal{N}_m^{tail}(\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \alpha_1, \alpha_2)$*, of two partial Gaussian distributions* $\mathcal{N}^{tail}(\mu_1, \sigma_1^2, \alpha_1)$ *and* $\mathcal{N}^{tail}(\mu_2, \sigma_2^2, \alpha_2)$*, as:*

$$\mathcal{N}_m^{tail}(\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \alpha_1, \alpha_2) =$$
$$\mathcal{N}^{tail}(\max(\mu_1, \mu_2), \max(\sigma_1^2, \sigma_2^2), \max(\mu_1, \mu_2) + \max(0, \alpha_1 - \mu_1, \alpha_2 - \mu_2))$$

In the following, we show that the merged partial Gaussian distribution is greater than each of the distributions used in the construction, as outlined in Theorem 10.6.1. We show this step-by-step, upper bounding each of the two partial Gaussian distributions until both reach the merged distribution. We provide a lemma and illustration for each step below.

Theorem 10.6.1. *The merged partial Gaussian distribution defined by* $\mathcal{N}_m^{tail}(\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \alpha_1, \alpha_1)$ *is an upper bound of each of the two distributions* $\mathcal{N}^{tail}(\mu_1, \sigma_1^2, \alpha_1)$ *and* $\mathcal{N}^{tail}(\mu_2, \sigma_2^2, \alpha_2)$*.*

In Lemma 10.6.2, we show that shifting the mean of a partial Gaussian distribution to a higher value while keeping the distance between the mean and the starting point α unchanged gives an upper bounding distribution. This is illustrated in Fig. 10.14.

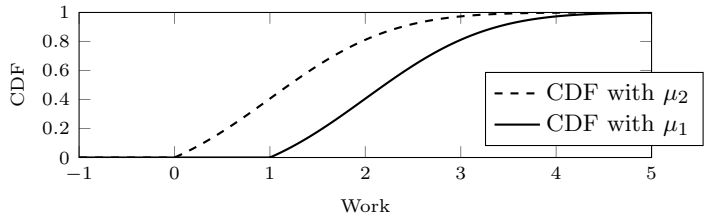


Fig. 10.14: CDFs of two partial Gaussian distributions as in Lemma 10.6.2. In this figure $\mu_1 = 2$, $\mu_2 = 1$, $\sigma^2 = 1$ and $\alpha_\Delta = -1$.

Lemma 10.6.2. *The partial Gaussian distribution* $\mathcal{N}^{tail}(\mu_1, \sigma^2, \mu_1 + \alpha_\Delta) \geq \mathcal{N}^{tail}(\mu_2, \sigma^2, \mu_2 + \alpha_\Delta)$ *if* $\mu_1 \geq \mu_2$.

Proof. From Definition 10.3.2, we know that the scaling factor of the partial Gaussian distribution depends only on α_Δ and σ^2 that are equal for the two distributions. From this we conclude that the CDF of $\mathcal{N}^{tail}(\mu_1, \sigma^2, \mu_1 + \alpha_\Delta)$ is the CDF of $\mathcal{N}^{tail}(\mu_2, \sigma^2, \mu_2 + \alpha_\Delta)$ translated $\mu_1 - \mu_2$ to the right. Therefore the CDF of $\mathcal{N}^{tail}(\mu_1, \sigma^2, \mu_1 + \alpha_\Delta)$ is always below that of $\mathcal{N}^{tail}(\mu_2, \sigma^2, \mu_2 + \alpha_\Delta)$. □

In Lemma 10.6.3, we show that increasing the variance to a higher value while keeping the mean and starting point unchanged gives an upper bounding distribution if the starting point α is at the mean or higher. This is illustrated in Fig. 10.15.

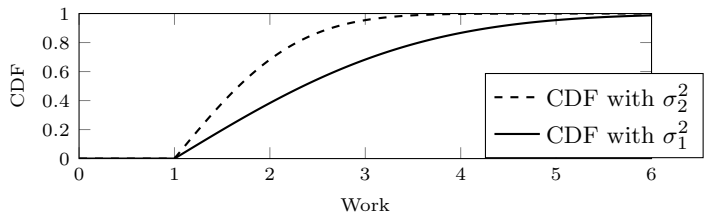


Fig. 10.15: CDFs of two partial Gaussian distributions as in Lemma 10.6.3. In this figure $\mu = 1$, $\sigma_1^2 = 4$, $\sigma_2^2 = 1$ and $\alpha = 1$.

Lemma 10.6.3. *The partial Gaussian distribution* $\mathcal{N}^{tail}(\mu, \sigma_1^2, \alpha) \geq \mathcal{N}^{tail}(\mu, \sigma_2^2, \alpha)$ *if* $\sigma_1^2 \geq \sigma_2^2$ *and* $\alpha \geq \mu$.

Proof. Let $\sigma_1^2 = k \cdot \sigma_2^2, k \geq 1$. Since the partial Gaussian functions are normalized to integrate to 1, the PDF of $\mathcal{N}^{tail}(\mu, \sigma_2^2, \alpha)$ at $x \geq \alpha$ can be written as $C_2 \cdot e^{-\frac{(x-\mu)^2}{2\sigma_2^2}}$, with C_2 as the normalization factor. Analogously we have the PDF of $\mathcal{N}^{tail}(\mu, \sigma_1^2, \alpha)$ as $C_1 \cdot e^{-\frac{(x-\mu)^2}{2k^2 \cdot \sigma_2^2}}$, with C_1 as the normalization factor. Let us evaluate the rate of decline in the PDFs between x and $x+\Delta x$, $\Delta x > 0$. Since $\alpha \geq \mu$ the PDF is declining. Dividing the PDF at x with the PDF at $x + \Delta x$ results in exponential functions with the coefficients $\frac{(\Delta x \cdot (\Delta x + 2(x-\mu))}{2\sigma_2^2}$ and $\frac{(\Delta x \cdot (\Delta x + 2(x-\mu))}{2k^2 \cdot \sigma_2^2}$ respectively. The PDF associated with $\sigma_1^2 = k \cdot \sigma_2^2$ has a lower rate of decrease than σ_2^2. This implies that the CDF associated with σ_2^2 has a more rapid growth from 0 and remains above the CDF associated with σ_1^2. □

With these lemmas in place, we can prove Theorem 10.6.1, restated here for convenience.

Theorem 10.6.7. *The merged partial Gaussian distribution defined by* $\mathcal{N}_m^{tail}(\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \alpha_1, \alpha_1)$ *is an upper bound of each of the two distributions* $\mathcal{N}^{tail}(\mu_1, \sigma_1^2, \alpha_1)$ *and* $\mathcal{N}^{tail}(\mu_2, \sigma_2^2, \alpha_2)$.

Proof. In the first step, we apply Lemma 10.6.2 and upper bound the execution times of the two distributions as:

$$\mathcal{N}^{tail}(\mu_1, \sigma_1^2, \alpha_1) \leq \mathcal{N}^{tail}(max(\mu_1, \mu_2), \sigma_1^2, max(\mu_1, \mu_2) + \alpha_1 - \mu_1)$$

and:

$$\mathcal{N}^{tail}(\mu_2, \sigma_2^2, \alpha_2) \leq \mathcal{N}^{tail}(max(\mu_1, \mu_2), \sigma_2^2, max(\mu_1, \mu_2) + \alpha_2 - \mu_2)$$

In a second step we apply Lemma 10.4.4 and derive upper bounds on the distributions as:

$$\mathcal{N}^{tail}(max(\mu_1, \mu_2), \sigma_1^2, max(\mu_1, \mu_2) + \alpha_1 - \mu_1) \leq$$
$$\mathcal{N}^{tail}(max(\mu_1, \mu_2), \sigma_1^2, max(\mu_1, \mu_2) + max(0, \alpha_1 - \mu_1, \alpha_2 - \mu_2))$$

and:

$$\mathcal{N}^{tail}(max(\mu_1,\mu_2),\sigma_2^2,max(\mu_1,\mu_2)+\alpha_2-\mu_2) \leq$$
$$\mathcal{N}^{tail}(max(\mu_1,\mu_2),\sigma_2^2,max(\mu_1,\mu_2)+max(0,\alpha_1-\mu_1,\alpha_2-\mu_2)$$

In a third step, we apply Lemma 10.6.3 to upper bound:

$$\mathcal{N}^{tail}(max(\mu_1,\mu_2),\sigma_1^2,max(\mu_1,\mu_2)+max(0,\alpha_1-\mu_1,\alpha_2-\mu_2)) \leq$$
$$\mathcal{N}^{tail}(max(\mu_1,\mu_2),max(\sigma_1^2,\sigma_2^2),max(\mu_1,\mu_2)+max(0,\alpha_1-\mu_1,\alpha_2-\mu_2))$$

and:

$$\mathcal{N}^{tail}(max(\mu_1,\mu_2),\sigma_2^2,max(\mu_1,\mu_2)+max(0,\alpha_1-\mu_1,\alpha_2-\mu_2)) \leq$$
$$\mathcal{N}^{tail}(max(\mu_1,\mu_2),max(\sigma_1^2,\sigma_2^2),max(\mu_1,\mu_2)+max(0,\alpha_1-\mu_1,\alpha_2-\mu_2))$$

This concludes our proof. □

10.6.3 Merging States in the Markov Model

Here, we describe how to merge two states in the modified execution time model. Without loss of generality, we describe how to merge the last two states, $S-1$ and S, to reduce the number of states from S to $S-1$. States can be reordered to merge any two states, and the process can be repeated to merge any number of states.

Recall that the *M* element $m_{a,b}$ represents the conditional probability of being in state b at task period $i+1$, given that at task period i, the state is a. Let $m_{a,b}$ represent an element in the transition matrix prior to merging and $m^m_{a,b}$ an element in the transition matrix after merging. In the new $(S-1)\times(S-1)$, the element values are calculated according to Eq. (10.40). All $m^m_{a,b}, a < S-1, b < S-1$ remain the same as $m_{a,b}$ because these are the transition probabilities of states unaffected by the merge. For $m^m_{a,S-1}, a < S-1$, that is the probability of moving from an unchanged state into the merged state, the transition probabilities into the merged states are summed. $m^m_{S-1,b}, b < S-1$ is the probability of moving from the merged state into an unchanged state. The transition probabilities from the merged state are weighted means of

the transition probabilities for the original states, weighted with the stationary probabilities. Finally, $m^m_{S-1,S-1}$ is the probability of staying in the merged state. For each of the merged states, we sum the probability of staying in the state or moving to the other of the merged states. A weighted mean is calculated for these sums with the stationary probabilities of the states.

$$m^m_{a,b} = \begin{cases} m_{a,b} & a < S-1, b < S-1 \\ m_{a,S-1} + m_{a,S} & a < S-1, b = S-1 \\ \frac{\xi(S-1)\cdot m_{S-1,b} + \xi(S)\cdot m_{S,b}}{\xi(S-1)+\xi(S)} & a = S-1, b < S-1 \\ \frac{\xi(S-1)\cdot(m_{S-1,S-1}+m_{S-1,S}) + \xi(S)\cdot(m_{S,S-1}+m_{S,S})}{\xi(S-1)+\xi(S)} & a = S-1, b = S-1 \end{cases} \tag{10.40}$$

In the merged Markov model, we have emission distributions $C_s \sim \mathcal{N}^{tail}(\mu_s, \sigma^2_s, \alpha_s), s < S-1$. For state $S-1$ the emission distribution is the merged partial Gaussian distribution $C_{S-1} \sim \mathcal{N}^{tail}_m(\mu_{S-1}, \mu_S, \sigma^2_{S-1}, \sigma^2_S, \alpha_{S-1}, \alpha_S)$.

In the merged Markov Model, transition probabilities remain unchanged, and emission distributions are unchanged or upper-bounded. The merged model is more pessimistic, and a DMP bound derived with the proposed method is safe. Probabilities of workload depletion are lower compared to the model prior to the merge. For jobs associated with a certain accumulation vector, the derived probability of deadline miss and the proportion of those jobs contributing carry-over into the next task period are the same or higher. The probability of job arrivals resulting in longer accumulation vectors, β is higher for the merged model compared to the original for the same number of accumulation periods N.

10.7 Evaluation

10.7.1 Goal of the Evaluation

The goal of the evaluation is to compare the obtained bounds with empirical deadline miss rates to verify that the method is applicable for a realistic use case, to see how the bound evolves with the workload accumulation iterations,

and to see how different server parameters and deadlines affect the pessimism. We compare to state of the art deadline miss probability estimates [26]. We also compare with simulation of the fitted Gaussian-emission Markov model, to evaluate the validity of this model for the use case, and to see the pessimism for a particular execution time state. Further, we show an estimate of the deadline miss probability assuming independence to see the effect of dependence in the use case.

10.7.2 Use Case and Test Setup

We evaluated the proposed deadline miss probability bound with a control task for a Furuta pendulum, a rotary inverted pendulum [57]. The control task implements a square root Kalman filter [38] estimating angles and angular velocities near the pendulum upright position and a PD controller for stabilizing the pendulum upright at angle 0 of the arm. A separate task simulates the pendulum dynamics and provides an asynchronous TCP server. The control task connects to the server to retrieve arm and pendulum angles and send the control signal. The control task runs periodically with a frequency of 500Hz. Tests were performed on a Raspberry Pi 3B+ with a `PREEMPT_RT`-patched version of Raspberry Pi OS. The control task was pinned to a core set up as an exclusive `cpuset` and scheduled with the Linux CBS implementation `SCHED_DEADLINE`. The simulator task was pinned to another core using `cpuset`. It runs periodically with the same frequency and was FIFO scheduled with the highest priority. The TCP server of the simulator runs in a separate thread. All cores were run with scaling governor `performance`. USB Ethernet and WiFi were disabled during the tests. The `ftrace` framework was used to record `sched` events and collect nanosecond-precision timestamps. The control task was scheduled with `SCHED_DEADLINE` setup with high bandwidth and a long server period, resulting in each job finishing within the server period. The time from the `sched_switch` event where the task is switched in to the event where it is switched out was taken as the execution time of a job. In some rare occasions there are several `sched_wakeup` events recorded close to each other in the same period. There are $50,011$ `sched_wakeup` events in the log from $50,000$ periods. One of these is due to an extra wake up when finishing the task after all periods, but 10 are due to preemptions by kernel space

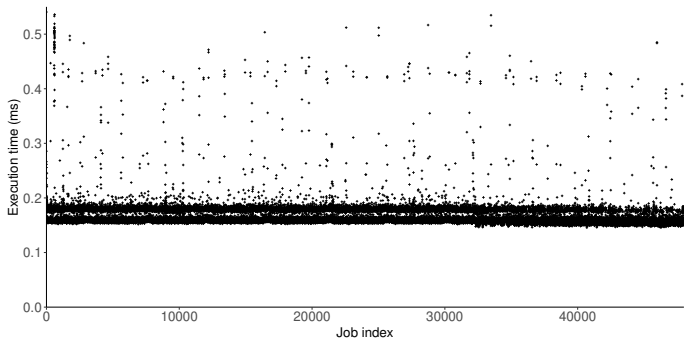


Fig. 10.16: The recorded execution time trace of the control task.

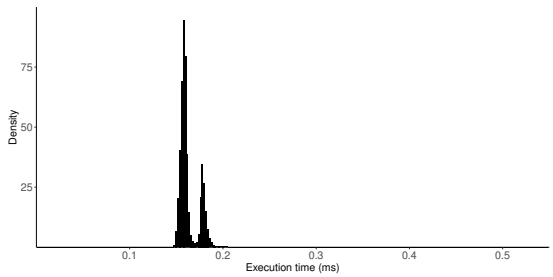


Fig. 10.17: The density distribution of the execution times starting at job 2 000.

tasks. In these cases, the execution time is taken as the sum of the time frames from switch in to switch out.

Recorded execution times from the control task running 50 000 periods are shown in Fig. 10.16. There was a run-in period with a higher proportion of execution times at 0.5ms at the beginning of the trace. The execution times of the first 2 000 jobs were discarded before fitting the HMM to the trace. The reason for this is that we want to perform the evaluation under the given assumptions. One assumption is stationarity, and therefore we exclude this part that appears to be a transient period. In Fig. 10.17, we display the density distribution of the execution time trace starting at job 2 000. The autocorrelation of the trace from job 2 000 is shown in Fig. 10.18.

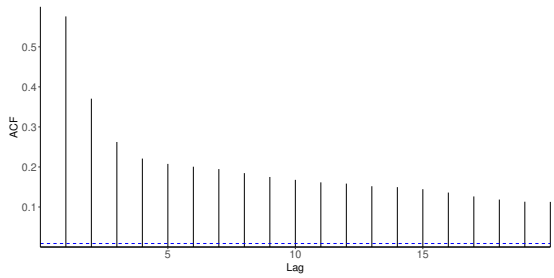


Fig. 10.18: The autocorrelation of the execution times sequence starting at job 2 000.

The evaluation was performed with three different configurations of server budget and period ratios:

1. $Q = 0.06$ ms, $n = 5$, $k_1 = 8$, $k_2 = 10$,
2. $Q = 0.07$ ms, $n = 4$, $k_1 = 6$, $k_2 = 8$, and
3. $Q = 0.08$ ms, $n = 4$, $k_1 = 6$, $k_2 = 8$.

Two relative deadlines were evaluated for each of these configurations.

The control task was scheduled with `SCHED_DEADLINE` configured with the different server budgets and period ratios. The task was configured with the relative deadline and logged the number of deadline misses in each 500-job-interval. During these tests, `sched` events are also recorded with the `ftrace` framework, to avoid that any introduced overhead by the tracing causes higher bounds and estimates compared to the empirical results. These recorded traces are not used further.

This small but realistic use case illustrates the method's applicability. Code and data related to the evaluation are available online[1].

10.7.3 Markov Model

The method outlined in Friebe *et al.* [29] was started with 10 initial states and identified an HMM with 8 states. The transition matrix is shown in Eq. (10.41),

[1] `https://github.com/annafriebe/ContMM_RT_BoundDMP`

and from this we conclude that the Markov Chain is irreducible. The resulting state means, standard deviations, and stationary probabilities are displayed in Table 10.3. The average computational requirement over a task period is about 0.164 ms, obtained from multiplying stationary probabilities with means and summing the products. The servers providing the lowest computational resource guarantee 0.28 ms computation time per task period, so all accumulation sequence Markov Chains are ergodic and we will have idle points in the server. The highest mean and standard deviation are observed in state 3. This state has a low stationary probability, only 0.7%, but the transition probability $m_{3,3}$ of staying in state 3 from one round to the next is as high as 63%. This dependence increases the DMP in state 3 and overall.

$$\begin{pmatrix} .739 & .051 & .002 & .003 & .162 & .001 & .041 & .001 \\ .056 & .350 & .012 & .000 & .523 & .008 & .051 & .000 \\ .000 & .310 & .633 & .003 & .000 & .044 & .010 & .000 \\ .006 & .000 & .002 & .408 & .004 & .054 & .000 & .526 \\ .000 & .038 & .002 & .003 & .834 & .001 & .121 & .000 \\ .000 & .000 & .004 & .681 & .063 & .225 & .000 & .028 \\ .377 & .011 & .001 & .000 & .500 & .000 & .107 & .003 \\ .009 & .001 & .002 & .296 & .000 & .038 & .001 & .654 \end{pmatrix} \tag{10.41}$$

10.7.4 Evaluated Methods

Six different methods were compared:

- **Linux-CBS** : Empirical deadline-miss ratio. The control task was scheduled with Linux `SCHED_DEADLINE` configured with each setting of server budget Q, task to server period ratio n and evaluated with the different relative deadline to server period ratios k. The task period was $2ms$ for all configurations, resulting in different bandwidths. 10 runs of the $50\,000$-job task were performed for each configuration. The empirical deadline miss ratio was calculated from deadline misses after the $2\,000$-job run-in period.

- **Sim-Cont**: A deadline-miss probability derived by generating execution times from the fitted HMM and feeding them into a CBS simulator with the

different server reservations, period ratios, and deadline configurations. A sequence of 10^6 samples was generated from the continuous-emission Markov model described by Table 10.3 and Eq. (10.41).

- **Ind**: A deadline-miss probability derived by assuming independent execution times, i.e., generating execution times by randomly sampling from the recorded trace and feeding them into a CBS simulator with the different server reservations, period ratios and deadline configurations. A sequence of 10^6 samples was generated.

- **PROSIT**: A deadline-miss probability derived with PROSITool [26]. A 6-state discrete-emission HMM is fitted to the execution time trace, using a 10 μs scaling factor for resampling. This HMM is evaluated with PROSIT's solver for steady-state deadline-miss probabilities with the different CBS configurations.

- **Bound-8**: A deadline-miss probability bound derived from the fitted 8-state continuous-emission Markov model and the methods in Section 10.4. The HMM is characterized by Table 10.3 and Eq. (10.41). The maximum number of accumulation periods was set to 10. The $\beta(s)_1$ for the first accumulation period were retrieved from the HMM simulation.

- **Bound-2**: A deadline-miss probability bound calculated according to Section 10.4 with a 2-state continuous-emission Markov model. The 2-state model was obtained from merging all states except State 3 from the 8-state model used for **Bound-8** as described in Section 10.6. The initial β values for the first accumulation period were retrieved from simulation with the merged 2-state model, and the maximum number of accumulation periods was set to 10.

Bound-8 and **Bound-2** are calculated as p_{dm} for each state according to Eq. (10.38), and the overall bound according to Eq. (10.39). The bounds for the state with the highest p_{dm} of **Bound-8** and **Bound-2** are compared to the deadline miss ratio of this state from **Sim-Cont**, as the empirical DMR **Linux-CBS** and **PROSIT** do not provide per-state estimates.

10.7.5 Results and Discussion

The bounds on the deadline miss probability p_{dm} derived along the workload accumulation for the 8-state model and the 2-state model are shown in Fig. 10.19, together with the average DMRs of the executions under `SCHED_DEADLINE`, deadline miss probability estimates from HMM simulation, assuming independence, and derived with PROSITool.

Deadline miss probabilities derived with HMM simulation and PROSITool are higher than empirical DMRs, except for the CBS configuration $0.08/4/8$. In this case, we observe p_{dm} of 0.021% derived with **Sim-Cont** and 0.020% with **PROSIT**, compared to 0.058% for the empirical **Linux-CBS**. This may be caused by a low probability state that is not captured in the fitting of the Markov Models. It may also be due to chance. This configuration has the lowest number of deadline misses, and a larger number of runs with **Linux-CBS** may have been needed for a reliable DMR estimate.

We observe that HMM simulation **Sim-Cont** estimates are consistently close to the **PROSIT** results, which indicates that the continuous emission distribution HMM is a valid approximation in the evaluated use case.

The resulting **Bound-8** bounds for the overall deadline miss probabilities are 1.76 to 10 times higher compared to HMM simulation **Sim-Cont**. The bounds for the state with the highest deadline miss probability are 1.3 to 4.1 times higher. Higher utilization and shorter relative deadlines give tighter bounds.

For the overall bounds of the merged model, **Bound-2**, they are 2.08 to 12.5 times higher than HMM simulation results **Sim-Cont**. Bounds for the state with the highest p_{dm} are 1.3 to 5.2 times higher compared to the simulation results.

The number of states and the scaling factor need to be provided when fitting an HMM in PROSIT, and the number of states and the scaling factor need to be provided. For this evaluation, several combinations of these parameters were tested. For 6 states and scaling factor 10 μs, 4 out of 6 states passed the PROSIT independence tests; this was the largest proportion found in the limited exploration. Some pessimism is introduced with PROSITool's resampling. Tighter or optimistic results were obtained with some of the fitted PROSIT models explored. The calculation time for PROSIT is greatly affected by the range of execution time values in the input trace and the scaling factor. For exam-

ple, taking the 6-state model and decreasing the scaling factor from 10 to 1 μs causes the computation time to increase from less than 0.5s to about 20 minutes on our platform, a factor of 3 000. The continuous approach has no resampling concept, and the calculation time is independent of the range of execution time values. A direct comparison between the proposed bound and PROSIT has not been performed. The proposed bound is implemented in Python and PROSIT in C++. PROSITool's computation time also varies a lot with the choice of scaling factor, and therefore, we assess that a direct comparison would not add much value to the evaluation.

In the different configurations, the time for the **Bound-2** and **Bound-8** calculations are logged for 5 and 10 accumulation periods, respectively. The means and standard deviations are shown in Table 10.4. The Python implementation of the **Bound-8** calculation for the 8-state model runs the first 5 accumulation periods in about one second and 10 accumulation periods in around one minute. With the 2-state bound **Bound-2**, the time required for an optimized implementation grows with the number of accumulation periods as $\mathcal{O}(N^2)$ instead of $\mathcal{O}(N^8)$ for 8 states. The non-optimized Python implementation of the bound calculation for the merged model runs in about 55 milliseconds for 10 accumulation periods.

In the evaluated use case, the tightest bound is already reached at 3-4 accumulation periods. Already at accumulation over 5 periods, the 2-state model is about 65 times faster than the 8-state model. At 10 accumulation periods, the 2-state model is more than 1000 times faster. Combining a low number of states with the use of accumulation vectors instead of accumulation sequences with ordering information provides a strong computational advantage.

Simulations and bounds of the state with the highest p_{dm} show results 50-100 times higher than the overall p_{dm}. While this should not be conflated with the Worst-Case Deadline Failure Probability, we believe that the concept of workload distribution per state is useful. In future work, we aim to develop the accumulation sequence approach relating to the probability of consecutive deadline misses.

In the evaluated use case, one state is identified with a much higher mean and variance than the others. It may be the case that this use case is especially well suited for state reduction into two states. Keeping the state with the highest

mean and merging the others may add pessimism in cases where states are more similar to the state with the highest mean.

10.8 Conclusions and Future Work

We have proposed a workload accumulation scheme starting from idle points to upper bound the deadline miss probability of a task. The task's computation times are described by a Markov Model with Gaussian emission distributions, and it is running on a reservation-based server.

A Markov model with Gaussian emission distributions allows for higher fitting process automation than discrete emission distributions, where the number of states and a scaling factor must be provided. Contrary to the discrete case, the time required to obtain the bound is independent of the range of execution times in the analyzed sequence, and no scaling factor is needed.

Further, we proposed a method for state merging. The bound computation time is reduced by reducing the number of states by merging. The time complexity for obtaining a bound for a model with S states considering N accumulation periods after an idle point is $\mathcal{O}(N^S)$. A bound is obtained early in the process and is updated successively.

The evaluation use case is a control task of a Furuta pendulum. The task is run with the Linux kernel implementation of CBS. The ratio of the number of missed deadlines to the total number of jobs is compared to the obtained bounds on the deadline miss probability. Bounds are derived from the fitted 8-state model and from a merged 2-state model obtained from the 8-state model. Furthermore, deadline miss probabilities for comparison are derived with a discrete emission-HMM [26, 25, 4], by simulation with the fitted HMM, and simulation assuming independence.

All bounds in the evaluation are higher than the simulation results. The overall bounds for the 8-state model are 1.76 to 10 times higher, and in the state with the highest deadline miss probability, the bounds are 1.3 to 4.1 times higher. The overall bounds obtained with the merged 2-state model are 2.08 to 12.5 times higher, and the bounds for the state with the highest deadline miss probability are 1.3 to 5.2 higher. All bounds are also safe compared to experimental deadline miss ratios. In the evaluation, the bound over 10 accu-

mulation periods takes about 0.06 seconds to calculate for the 2-state model, but a minute for the 8-state model, an improvement of a factor 1000. Combining the workload accumulation method with state number reduction by merging gives a strong computational benefit.

In future work, it would be interesting to develop the workload accumulation approach to evaluate the probability of consecutive deadline misses, or extend the approach to support DAG-based tasks. The bounds could potentially be evaluated for use in an adaptive setting to monitor changes in the deadline miss probability to adapt the Quality-of-Service (QoS) level.

Table 10.2: Overview of notation used in this paper.

Symbol	Description
	Basic notation
T	Task period
J_i	Job at task period i
a_i	Arrival time of J_i
d_i	Absolute deadline of J_i
D	Relative deadline
P	Server period
Q	Server budget
n	Number of server periods in a task period
k	Number of server periods in a relative deadline
S	Number of Markov states
M	State transition matrix
N	Number of task periods in workload accumulation
	Values of random variables
c_i	Execution time of J_i
f_i	Finishing time of J_i
v_i	Workload at task period i
h	Accumulation sequence of state visits in Markov chain since workload depletion
$\tilde{h}$	Accumulation vector of the number of visits in each Markov state since workload depletion
	Probability distributions and probabilities
C	Execution time distribution
$\mathcal{V}_h, \mathcal{V}_{\tilde{h}}$	Workload distribution associated with an accumulation sequence or vector
$m_{i,j}$	Transition probability from state i to state j
$\xi(s)$	Stationary probability of being in s
$p_{in}(s, \tilde{h})$	Probability of entering s with $\tilde{h}$
$p_{co}(s, \tilde{h})$	Probability that $\tilde{h}$ in s carries workload to the next task period
$p_{wd}(s)$	Probability of workload depletion in s
p_{dm}	Deadline miss probability
$\beta(s)_N$	Probability of being in state s with h longer than N.

Table 10.3: Means, standard deviation, and stationary probabilities of the fitted HMM states.

State number	1	2	3	4	5	6	7	8
Mean (ms)	0.178	0.178	0.323	0.158	0.159	0.169	0.181	0.153
Standard deviation (ms)	0.002	0.012	0.091	0.003	0.002	0.007	0.003	0.002
Stationary probability	0.128	0.045	0.007	0.086	0.509	0.014	0.078	0.133

Table 10.4: Time of the bound calculations with the 2-state and 8-state models for the 6 configurations over 5 and 10 accumulation periods.

Bound calculation time	Mean (s)	Standard deviation (s)
Bound-2, 5 Accum. periods	0.0153	$4.88 \cdot 10^{-4}$
Bound-2, 10 Accum. periods	0.0552	$4.10 \cdot 10^{-3}$
Bound-8, 5 Accum. periods	1.00	0.0129
Bound-8, 10 Accum. periods	60.66	4.35

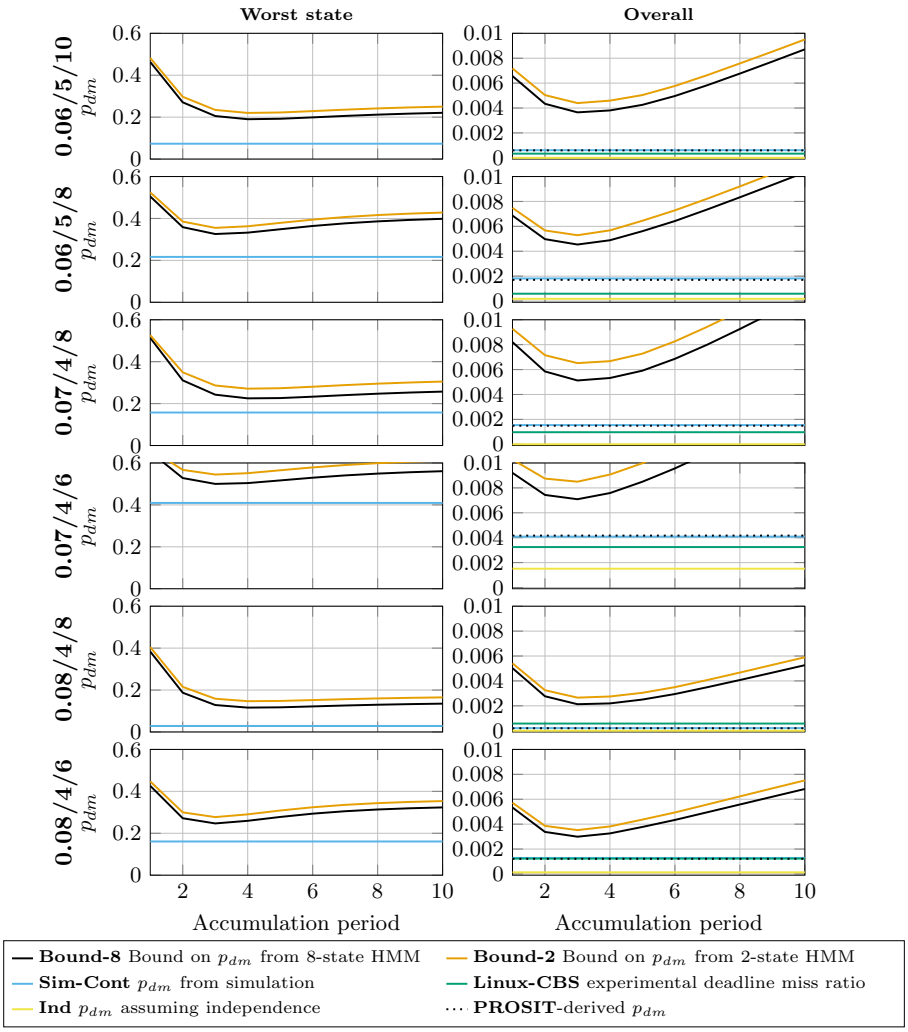


Fig. 10.19: Evolution over 10 accumulation periods of the **Bound-8** and **Bound-2** according to Eq. (10.38) for the worst state, and according to Eq. (10.39) for the task overall, compared to the other methods listed in Section 10.7.4.

Bibliography

[1] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 4–13, 1998.

[2] Luca Abeni and Giorgio Buttazzo. QoS guarantee using probabilistic deadlines. In *Euromicro Conf. Real-Time Syst. (ECRTS)*, pages 242–249, 1999.

[3] Luca Abeni and Giorgio Buttazzo. Stochastic analysis of a reservation based system. In *Int. Workshop on Par. and Distr. Real-Time Syst.*, volume 1, 2001.

[4] Luca Abeni, Daniele Fontanelli, Luigi Palopoli, and Bernardo Villalba Frías. A Markovian model for the computation time of real-time applications. In *IEEE Instrum. & Meas. Tech. Conf. (I2MTC)*, pages 1–6, 2017.

[5] Luca Abeni, Nicola Manica, and Luigi Palopoli. Efficient and robust probabilistic guarantees for real-time tasks. *J. of Syst. and Soft.*, 85(5):1147–1156, 2012.

[6] Benny Åkesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I Davis. A comprehensive survey of industry practice in real-time systems. *Real-Time Systems*, 58(3):358–398, 2022.

[7] Guillem Bernat, Alan Burns, and Martin Newby. Probabilistic timing analysis: An approach using copulas. *Journal of Embedded Computing*, 1(2):179–194, 2005.

[8] Sergey Bozhko, Filip Marković, Georg von der Brüggen, and Björn B Brandenburg. What Really is pWCET? A Rigorous Axiomatic Proposal. In *IEEE Real-Time Systems Symposium (RTSS)*, 2023.

[9] Sergey Bozhko, Georg von der Brüggen, and Björn Brandenburg. Monte Carlo response-time analysis. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 342–355, 2021.

[10] Giorgio C Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems: Predictability vs Efficiency*. Springer, 2005.

[11] Jian-Jia Chen, Mario Günzel, Peter Bella, Georg von der Brüggen, and Kuan-Hsun Chen. Dawn of the dead (line misses): Impact of job dismiss on the deadline miss rate. *arXiv preprint arXiv:2401.15503*, 2024.

[12] Kuan-Hsun Chen, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. Critical instant for probabilistic timing guarantees: Refuted and revisited. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 145–157. IEEE, 2022.

[13] Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. Efficient computation of deadline-miss probability and potential pitfalls. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'19), March 25-29, Florence, Italy*, pages 896–901. IEEE, 2019.

[14] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *SIGCOMM Comput. Commun. Rev.*, 22(4):14–26, 1992.

[15] Stuart Coles. *An Introduction to Statistical Modeling of Extreme Values*, volume 208. Springer-Verlag, London, UK, 2001.

[16] Liliana Cucu-Grosjean. Independence-a misunderstood property of and for probabilistic real-time systems. *In Real-Time Systems: the past, the present and the future*, pages 29–37, 2013.

[17] Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J Cazorla. Measurement-Based Probabilistic Timing Analysis for multi-path programs. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 91–101, 2012.

[18] Robert I Davis, Alan Burns, and David Griffin. On the meaning of pWCET distributions and their use in schedulability analysis. In *In Proceedings Real-Time Scheduling Open Problems Seminar at (ECRTS'17)*, 2017.

[19] Robert I Davis and Liliana Cucu-Grosjean. A survey of probabilistic timing analysis techniques for Real-Time systems. *Leibniz Trans. Emb. Syst.*, 6(1):03–1–03:60, 2019.

[20] Robert Ian Davis and Liliana Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for Real-Time systems. *LITES: Leibniz Transactions on Embedded Systems*, pages 1–53, 2019.

[21] Jamile de Barros Vasconcelos and George Lima. Possible risks with EVT-based timing analysis: an experimental study on a multi-core platform. In *2022 XII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–8. IEEE, 2022.

[22] José Luis Díaz, Daniel F García, Kanghee Kim, Chang-Gun Lee, L Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 289–300, 2002.

[23] Jose Luis Diaz, Jose Maria Lopez, Manuel Garcia, Antonio M Campos, Kanghee Kim, and Lucia Lo Bello. Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *IEEE Int. Real-Time Syst. Symp. (RTSS)*, pages 197–207, 2004.

[24] Bernardo Villalba Frías. *Bringing Probabilistic Real-Time Guarantees to the Real World*. PhD thesis, University of Trento, 2018.

[25] Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. Probabilistic real-time guarantees: There is life beyond the iid assumption. In *IEEE Real-Time and Embedded Tech. and Appl. Symp. (RTAS)*, pages 175–186, 2017.

[26] Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. The PROSIT tool: Toward the optimal design of probabilistic soft real-time systems. *Software: Practice and Experience*, 48(11):1940–1967, 2018.

[27] Anna Friebe, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte. Adaptive runtime estimate of task execution times using Bayesian modeling. In *IEEE Int. Conf. Emb. and Real-Time Comp. Syst. and Appl. (RTCSA)*, pages 1–10, 2021.

[28] Anna Friebe, Filip Marković, Alessandro Vittorio Papadopoulos, and Thomas Nolte. Continuous-emission Markov models for real-time applications: Bounding deadline miss probabilities. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 14–26, 2023.

[29] Anna Friebe, Alessandro V Papadopoulos, and Thomas Nolte. Identification and validation of Markov models with continuous emission distributions for execution times. In *IEEE Int. Conf. on Emb. and Real-Time Comp. Syst. and Appl. (RTCSA)*, pages 1–10, 2020.

[30] Mario Günzel, Niklas Ueter, Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. Probabilistic reaction time analysis. *ACM Transactions on Embedded Computing Systems*, 22(5s):1–22, 2023.

[31] Mor Harchol-Balter. *Introduction to probability for computing*. Cambridge University Press, Cambridge, United Kingdom ; New York, NY, USA, 1st ed. edition, 2024.

[32] Matthias Ivers and Rolf Ernst. Probabilistic network loads with dependencies and the effect on queue sojourn times. In *Int. Conf. Heterogeneous Netw. for Qual., Reliab., Sec. and Robust. (QShine)*, pages 280–296, 2009.

[33] M. Ross Leadbetter, Georg Lindgren, and Holger Rootzén. Conditions for the convergence in distribution of maxima of stationary normal processes. *Stoch. Proc. and their Appl.*, 8(2):131–139, 1978.

[34] Juri Lelli, Claudio Scordino, Luca Abeni, and Dario Faggioli. Deadline scheduling in the linux kernel. *Software: Practice and Experience*, 46(6):821–839, 2016.

[35] George Lima and Iain Bate. Valid application of EVT in timing analysis by randomising execution time measurements. In *23rd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17), April 18-21, Pittsburg, PA, USA*, pages 187–198. IEEE, 2017.

[36] George Lima, Dario Dias, and Edna Barros. Extreme value theory for estimating task execution time bounds: A careful look. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 200–211. IEEE, 2016.

[37] Rui Liu, Alex F Mills, and James H Anderson. Independence thresholds: Balancing tractability and practicality in soft real-time stochastic analysis. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 314–323, 2014.

[38] Lennart Ljung. *System identification : theory for the user*. Prentice-Hall information and system sciences series. Prentice Hall, Upper Saddle River, N.J, 2. ed. edition, cop. 1999.

[39] Yue Lu, Thomas Nolte, Iain Bate, and Liliana Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 351–362, 2012.

[40] Yue Lu, Thomas Nolte, Johan Kraft, and Christer Norström. A statistical approach to response-time analysis of complex embedded real-time systems. In *IEEE Int. Conf. Emb. and Real-Time Comp. Syst. and Appl. (RTCSA)*, pages 153–160, 2010.

[41] Yue Lu, Thomas Nolte, Johan Kraft, and Christer Norström. Statistical-based response-time analysis of systems with execution dependencies be-

tween tasks. In *IEEE Int. Conf. Eng. of Compl. Comp. Syst. (ICECCS)*, pages 169–179, 2010.

[42] Nicola Manica, Luigi Palopoli, and Luca Abeni. Numerically efficient probabilistic guarantees for resource reservations. In *IEEE Int. Conf. Emerg. Tech. & Factory Autom. (ETFA)*, pages 1–8, 2012.

[43] Filip Marković, Jan Carlson, Radu Dobrin, Bjorn Lisper, and Abhilash Thekkilakattil. Probabilistic response time analysis for fixed preemption point selection. In *13th IEEE International Symposium on Industrial Embedded Systems (SIES'18), June 6-8, Graz, Austria*, pages 1–10. IEEE, 2018.

[44] Filip Marković, Alessandro Vittorio Papadopoulos, and Thomas Nolte. On the convolution efficiency for probabilistic analysis of real-time systems. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[45] Filip Marković, Pierre Roux, Sergey Bozhko, Alessandro V Papadopoulos, and Björn B Brandenburg. CTA: A correlation-tolerant analysis of the deadline-failure probability of dependent tasks. In *Proceedings of the 44th IEEE Real-Time Systems Symposium (RTSS)*, 2023.

[46] Filip Marković, Thomas Nolte, and Alessandro Vittorio Papadopoulos. Analytical approximations in probabilistic analysis of real-time systems. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 158–171, 2022.

[47] Pau Martí, Josep M Fuertes, Gerhard Fohler, and Krithi Ramamritham. Improving Quality-of-Control using flexible timing constraints: metric and scheduling. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 91–100, 2002.

[48] J. (Jyotiprasad) Medhi. *Stochastic models in queueing theory*. Mathematics in science and engineering. Academic Press, Amsterdam ;, 2nd ed. edition, 2003.

[49] Alex F Mills and James H Anderson. A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. In *IEEE Int. Conf. Emb. and Real-Time Comp. Syst. and Appl. (RTCSA)*, pages 207–217, 2011.

[50] Luigi Palopoli, Daniele Fontanelli, Luca Abeni, and Bernardo Villalba Frías. An analytical solution for probabilistic guarantees of reservation based soft real-time systems. *IEEE Trans. Par. and Distr. Syst.*, 27(3):640–653, 2016.

[51] Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[52] Luca Santinelli, Jérôme Morio, Guillaume Dufour, and Damien Jacquemart. On the sustainability of the extreme value theory for WCET estimation. In *Int. W. on Worst-Case Exec. Time Anal.*, 2014.

[53] Moshe Shaked. *Stochastic orders*. Springer series in statistics. Springer, New York, 2007.

[54] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings Real-Time Technology and Applications Symposium*, pages 164–173, 1995.

[55] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. Efficiently approximating the probability of deadline misses in real-time systems. In *30th Euromicro Conference on Real-Time Systems (ECRTS'18), July 3-6, Barcelona, Spain*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[56] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, Katharina Morik, and Björn B Brandenburg. Efficiently approximating the Worst-Case Deadline Failure Probability under EDF. In *IEEE Real-Time Syst. Symp. (RTSS)*, pages 214–226, 2021.

[57] Nils Vreman, Anton Cervin, and Martina Maggio. Stability and performance analysis of control systems subject to bursts of deadline misses. In *33rd Euromicro Conf. Real-Time Systems (ECRTS 2021)*, 2021.

[58] Kevin Zagalo, Yasmina Abdeddaim, Avner Bar-Hen, and Liliana Cucu-Grosjean. Response time stochastic analysis for fixed-priority stable real-time systems. *IEEE Transactions on Computers*, 72(1):3–14, 2022.

Chapter 11

Paper D
Nip it in the Bud:
Job Acceptance Multi-Server.

Anna Friebe, Tommaso Cucinotta, Filip Marković,
Alessandro V. Papadopoulos, and Thomas Nolte.
Accepted at the 31st IEEE Real-Time and Embedded Technology and Applications Symposium, 2025.

Abstract

Computationally demanding tasks with highly variable execution times may require parallel processing. Scheduling such tasks with low deadline miss rates but without significant overprovisioning is challenging. This issue arises in applications like nonlinear optimization for Model Predictive Control (MPC). The Constant Bandwidth Server (CBS) provides timing isolation, supporting both hard and soft real-time tasks. However, scheduling parallel, time-varying jobs across multiple CBS instances requires static job-to-server assignments, which can lead to resource underutilization due to queued jobs awaiting specific servers. This paper introduces the Job Acceptance Multi-Server (JAMS), a mechanism in which multiple CBS instances share a common job queue, enabling flexible job dispatching for parallel workloads. JAMS incorporates a job dismissal mechanism to address overloads, ensuring that only jobs with guaranteed resource availability are accepted. Each CBS instance checks if it can complete a job by its deadline, given probabilistic knowledge on its execution times, dismissing unfeasible jobs to avoid excessive tardiness across queued tasks. Implemented in Linux, JAMS is evaluated with computation times drawn from an MPC task and synthetic datasets. The extensive experimental results we provide, demonstrate that JAMS effectively controls the deadline miss rate, maintaining it below a specified design threshold.

11.1 Introduction

In real-time systems, computational workloads can often be decomposed into smaller tasks that execute either in parallel or sequentially. Certain computations require strict sequential execution, but multiple such computations may be active concurrently, operating on different data. For instance, in Model-Predictive Control (MPC), an optimization problem is solved to determine a state trajectory and control commands over a defined horizon, with a time-varying convergence, particularly in nonlinear systems. As shown in [54], such tasks for different time windows can be processed in parallel on predicted states, updating results when states become available. Methods based on particle filters such as FastSLAM are often parallelized [5, 26] and may use an adaptive number of particles, resulting in varying computation times [22]. A state estimate can be obtained from a smaller number of particles although it may be less accurate [22]. Allocating computational resources to a collective group of tasks with a unified objective rather than to individual tasks enables efficient resource utilization..

The Constant Bandwidth Server (CBS) [2] provides timing isolation and supports integration of both soft and hard real-time tasks. In multicore systems, CBS instances may be allowed to migrate across cores, although each instance's utilization is capped at 1 [9]. Thus, CBS alone is not sufficient for resource assignment to a group of parallel tasks working toward a combined outcome, i.e. a thread pool.

To address this limitation, we introduce the Job Acceptance Multi-Server (JAMS), where multiple CBS instances share a global job queue, and queued jobs are dispatched when a server becomes available.

If a task in a server misbehaves and requires more computation resources than assumed at design time, the server guarantees that tasks outside the server still receive the required resources. However, job tardiness within the server may grow unboundedly, potentially disrupting the system functionality even when other tasks have sufficient computational resources. In such cases, JAMS utilizes a job dismissal mechanism to prioritize jobs with a high likelihood of meeting their deadlines, mitigating overload situations and allowing for smoother system recovery.

Job dismissal, used in some schedulability and response time analysis of real-time systems [32, 35], assumes that jobs are dismissed upon deadline misses or at another specified dismissal point, enhancing analytical tractability and mitigating adverse effects from misbehaving tasks or inaccuracies in execution time estimates. For instance, if the analysis relies on a Worst Case Execution Time (WCET) that was erroneously estimated, dismissing a job once it consumed its erroneously considered WCET at analysis and admission time, ensures that the analysis still holds for the jobs that behave correctly.

Despite these advantages of job dismissals, many real-time schedulers do not provide such a feature. One alternative is to implement in-task job abortion [37], or through dedicated programming abstractions, such as the *deadline exception* introduced in [18]. While this is often a suitable alternative that allows for tailor-made abortion points and can be implemented in addition to the proposed JAMS mechanism, it requires that the tasks are bug-free and non-compromised. Tasks cannot consider the total load of the system in the dismissal decision. Furthermore, dismissing a job at the time when it is about to start execution, rather than at a dismissal point during execution, frees resources for other jobs. To address these challenges, JAMS ensures that jobs dispatched to a server are guaranteed a certain amount of computation time prior to their deadline, by dismissing tardy jobs if needed..

Contribution: The contribution of this paper is twofold:

1. The JAMS framework, which enables a group of CBS servers to pick jobs from a global shared queue, facilitating flexible dispatching of jobs with varying computation times and a joint goal.

2. A job acceptance policy that dispatches jobs for processing only if there are resources guaranteeing a high probability for them to meet their deadlines, dismissing jobs queued beyond acceptable queue times.

11.2 Related Work

Task models for parallel workloads include the fork-join model [33], where a main thread executes sequentially up until a point where it forks into a number of threads that execute a parallel part of the computation. This is followed by a

synchronization where the parallel threads are terminated and the main thread continues. An extension is the parallel synchronous task model [43], where parallel computational segments may be consecutive, and two segments may have different number of threads. Lupu and Goossens [34] presented a multi-thread periodic task model, where each task periodically generates a number of subprograms (threads). Hard real-time fixed-priority schedulability tests for constrained deadlines are provided. In the parallel MPC example from [54], optimizations start periodically, but they overlap, so there are no points when threads simultaneously synchronize and join.

Another common parallel task model is the gang task model, where a number of threads are required to run concurrently because they interact. Coscheduling of such processing working sets was introduced by Ousterhout [39]. A one-gang-at-a-time policy [7] has been proposed to reduce interference and turn the multicore parallel scheduling into an equivalent of uniprocessor scheduling. Recently, soft real-time scheduling of gang tasks has been considered [6], including presenting server-based scheduling policies and schedulability tests. For the tasks in this paper, we consider functionality that may be run in parallel, but there is no interaction or need for coscheduling, so the gang task model is unnecessarily restrictive.

Scheduling of a set of tasks with multiprocessor bandwidth reservations has been implemented to support hierarchical scheduling with bounded delay [40]. A similar approach has been taken to support real-time containers [1]. Both these approaches use control groups to separate task sets and modify the Linux CBS implementation `SCHED_DEADLINE` to schedule the root level of a hierarchical schedule, while the lower level is scheduled with fixed priority. In JAMS, CBS is used for the low-level scheduling.

For systems without job dismissal, it is clear that the average requested computational resources must be lower than the average provided resource for these systems to be stable [20]. In such a stable system, there exist points in time when the job queue is empty [55]. In a system where jobs are discarded, stability can be achieved without this requirement, but some of the requested computational resources may be rejected due to the discarding policy. Jobs may also be delayed or discarded before they start due to a lack of resources, scheduling priorities, and discarding policy. Chen *et al.* [16] compared the ef-

fect on the deadline miss rate of varying the dismiss point after the deadline for a uniprocessor system where task computation times were independent discrete random variables and the average resource demand was lower than the supply. While not directly applicable to our multiprocessor use case, potentially with overload and correlated computation times, a later dismiss point resulted in a higher deadline miss rate, but the rate converged. Manolache *et al.* [35] analyzed task graphs with stochastic computation times with an upper bound on the number of concurrently active instantiations of each task graph. They concluded that denying service to a newly arrived task graph considerably reduces the number of states and schedulability analysis time compared to rejecting the oldest instantiation in the system. Pazzaglia *et al.* [41] compared the effect on control robustness of different strategies of handling deadline misses, including different dismissal policies. For these control applications, killing a job at the deadline or skipping the next job if a deadline has been missed were both preferable to queueing jobs.

In the area of mixed criticality systems, LO-criticality jobs are aborted or dismissed to ensure that HI-criticality jobs meet their deadlines [14, 10, 27]. One of the criticisms from system engineers discussed in [14, 11] of the assumptions of Mixed-criticality systems is that LO-criticality jobs should receive some service if at all possible. In a resilient system [14] started jobs run to completion, and a task is considered robust if it can safely drop one *non-started* job in any extended time interval [13]. In the bailout protocol [11], HI-criticality jobs that overrun their budget continue their execution. The overrun is compensated for by not starting LO-criticality jobs, and by accounting for jobs that use less resources than budgeted until the system has returned to normal execution mode. In [31] HI-criticality jobs were monitored in a multiprocessor system, and if one such job risked exceeding its isolation-based WCET, concurrently running LO-criticality jobs were suspended to ensure they didn't interfere.

In the Robust Earliest Deadline algorithm [15] EDF-scheduled tasks in a uniprocessor system are associated with values, deadline tolerances and a criticality level. If a job arrival leads to a WCET-based overload situation, non-critical jobs with low value are rejected from the ready queue. Rejected jobs may be recovered if jobs complete early. Due to the preemptive scheduling, partly computed jobs may be aborted. In [45], this approach was applied to

aperiodic jobs in a Total Bandwidth Server (TBS). It has been pointed out that QoS guarantees for individual tasks cannot be provided in such a case, but a separate server for each task is required [3].

In [12] weakly hard real-time systems were introduced, enabling specification of a minimum number of met or consecutively met deadlines in every window of a specified number of task invocations, or a maximum number of consecutive deadline misses. These are alternative or complementary to the probabilistic approach.

Tong *et al.* [47] proposed holistic budgeting of Directed Acyclic Graph (DAG) tasks to decrease DAG drop rates. The slack from nodes that complete early and from nodes that cannot start due to overruns in predecessor nodes is used to complete overrunning nodes. This leads to a lower drop rate than dropping the DAG when one node overruns, showing the advantages of holistic budgeting and scheduling of potentially parallel work. In queuing theory, a significant amount of work analyzes queues with reneging or customer abandonment. Kruk *et al.* [32] analyzed EDF queues where jobs are dismissed at their deadline. Ward [49] surveyed results for queues with reneging. Reneging can refer to client abandonment, where a client (job) leaves the queue according to a probability distribution over the waiting time or reneging when a deadline is met, or a buffer is full. Ward concludes that for the overloaded many-server case, the story is complete. Towsley and Panwar [48] introduced Stochastic Earliest Deadline policies in the case where deadlines are not known to the scheduler, and analyzed the finite buffer case. If a job arrives at a full buffer, the policy of removing the job stochastically closest to its deadline is at least as good as the arbitrary policy. Whitt [50] investigates overloaded queues with different abandonment time and service time distributions. Abandonment time distributions significantly affect the mean queue length and waiting time. Most work considers abandonment distributions that depend on job waiting times, but there is also admission control or buffer overflow management that takes into account the state of the queue as a whole. More recently, Whitt has reviewed work on time-varying queues [52]. In [51] Whitt studied service rate controls to stabilize queue performance with time-varying arrival rate. Performance measures were mean queue lengths and mean waiting times. It was shown that any control that asymptotically stabilizes mean queue lengths cannot also stabilize

mean waiting times.

In networking, packets sometimes need to be dropped due to full buffers. In Active Queue Management, packets are dropped preemptively before the buffer is full, to reduce congestion and waiting times. Random Early Detection [21] has been proposed for congestion avoidance, where packets are dropped with a probabilistic approach. More recently, CoDel has been proposed [38], that is based on the minimum waiting time in the queue over a specified time interval. If this minimum waiting time exceeds a target value, packets start to be dismissed. The time between successive dismissals is decreased until the target waiting time is met.

11.3 System Model and Notation

11.3.1 Task Model

We consider a task τ that releases at most κ jobs at the same instant. Job release instants are separated by at least a minimum separation time p. Each job J_j has the arrival time a_j. A job computation time c_j is an outcome of the random variable $\mathcal{C}$; thus, the job's finishing time f_j and response time are outcomes of random variables as well. We denote the response time random variable $\mathcal{R}$. The task has a relative deadline D, so each job has the deadline $a_j + D$. The job computation times are upper bounded by the WCET $c^{\uparrow}$.

The computation time quantile c_ϕ is defined as:

$$c_\phi := \inf \{x : \mathbb{P}[\mathcal{C} \leq x] \geq \phi\} \tag{11.1}$$

The random variable $\mathcal{W}$ is the waiting time of an arriving job. The random variable $\mathcal{B}$ is the remaining work of serviced jobs, and the random variable $\mathcal{L}$ is the work of all queued jobs. The work arriving in an interval of length Δ is a random variable denoted by $\Upsilon(\Delta)$. $\mathcal{L}$ and $\Upsilon(\Delta)$ are sums of computation time random variables, and $\mathcal{B}$ is upper bounded by such a sum.

The notation used in the paper is outlined in Table 11.1.

11.3.2 CBS Background

We recall the CBS [4] operation. A real-time task τ is scheduled by an associated CBS S_i, described by its static parameters: maximum budget Q_i and period T_i, resulting in a utilization or bandwidth of $U_i = \frac{Q_i}{T_i}$. S_i also keeps track at run-time of the remaining budget q_i and the absolute deadline d_i, two dynamic quantities that change with the current time $t \in \mathbb{N}$. Let δt denote the length of a generic time interval in which (a job of) a server has been executed on a CPU. S_i is *idle* at time t, if τ has no pending jobs at t, that is if at time t, there exists no job J_j such that $a_j < t < f_j$. If S_i is not *idle* at time t, it is *active* if it has remaining budget ($q_i > 0$). If the budget is depleted, it is *recharging*. A flowchart of the runtime changes of the server is shown in Fig. 11.1. For the purpose of this paper, each server runs on a specified processor. The total utilization of the processor of S_i is denoted by $U_{i\Sigma}$. We assume partitioned EDF scheduling of servers, flanked with a per-CPU utilization-based test, which ensures each CBS gets its reserved budget within its deadline

11.3.3 Scheduling Parallel Workload with a Group of CBS

The task τ is scheduled by a JAMS to support a potentially parallel workload. n CBS instances, each with server period T and maximum budget Q, are set up to share the same job queue, as illustrated in Fig. 11.2. Essentially JAMS implements a thread pool of n worker threads with guaranteed processing capacity, along with an interface to submit work to be processed, and retrieve information about the work's status and the result if it is ready. The operation of JAMS is described in Section 11.5.

11.3.4 Definition and Assumptions

We define the concept of JAMS overload in a time interval:

Definition 11.3.1. *A JAMS is overloaded in the interval $[t_0, t_1]$ if the total computation times of arriving jobs in the interval exceeds the expected provided resource by the servers, that is if $\frac{n \cdot Q \cdot (t_1 - t_0)}{T} < \sum_{j | t_0 \leq a_j \leq t_1} c_j$.*

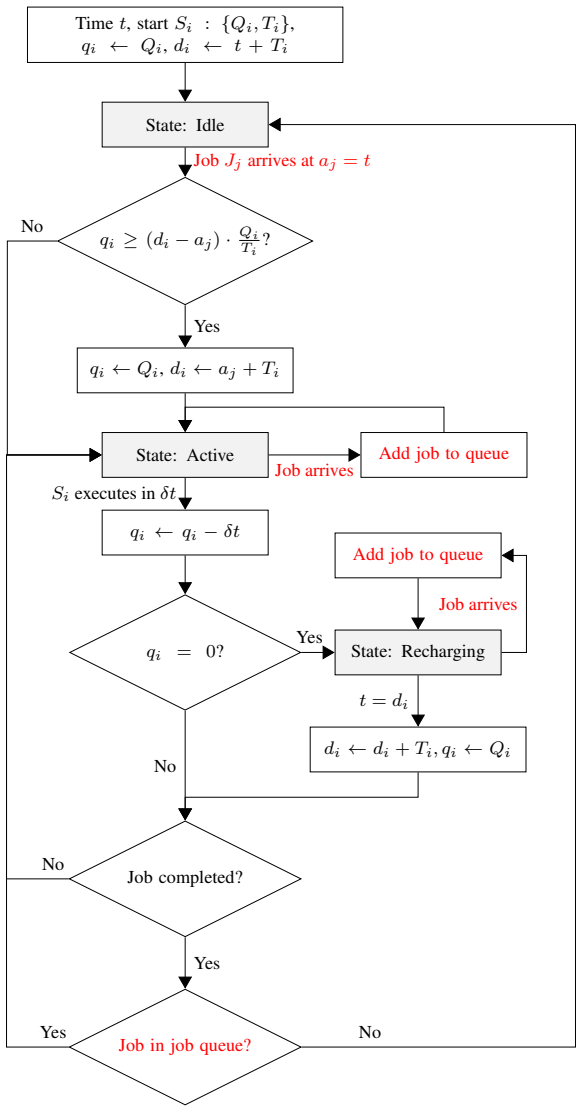


Fig. 11.1: Flowchart of the CBS update at runtime. The job handling to be updated by the JAMS is marked in red.

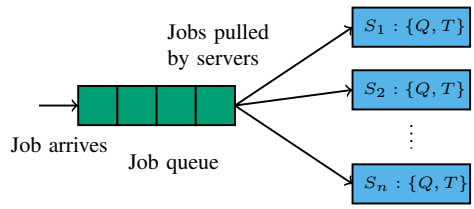


Fig. 11.2: Overview of JAMS.

In the remainder of the paper, we make the assumptions outlined below. Asm. 11.3.1 indicates that for intervals longer than Δ_B, individual job computation times become negligible relative to the total workload, and the amount of arriving work in the interval is characterized by a work arrival rate bound ρ. Asm. 11.3.2 implies that a job arriving when an idle server has maximum budget must be accepted.

Assumption 11.3.1. *For time intervals longer than Δ_B, the amount of arriving work $\Upsilon(\Delta)$ is bounded by a work arrival rate bound ρ and a probability ϵ, such that $\mathbb{P}\left[\Upsilon(\Delta) > \rho \cdot \Delta\right] < \epsilon, \Delta > \Delta_B$.*

Assumption 11.3.2. *The server configuration allows for completing a job with the computation time quantile within the task's deadline, that is $c_\phi \leq Q \cdot \left\lfloor \frac{D}{T} \right\rfloor$.*

11.4 Motivating Examples and Problem Formulation

Example 11.4.1. *A task τ releases one job J_j every $p = 20$. The result of J_j is expected at latest at $a_j + 60$, we have $D = 60$. There is a fallback option in the case of occasional deadline misses. The computation time of a job J_j is a random variable, that takes the value of 20 with probability 0.9 and 38 with probability 0.1, the probability mass function is $(\mathbb{P}\left[\mathcal{C} = 20\right] = 0.9, \mathbb{P}\left[\mathcal{C} = 38\right] = 0.1)$. The computation time random variables are independent and identically distributed (i.i.d.).*

For this example, an average of 21.8 units of work arrive every 20 time units. Assuming that each processor provides 20 computational units in each

period, the task in Example 11.4.1 cannot be scheduled on a single processor or in a single CBS.

When scheduling τ, we have some options. We may split τ into s tasks τ_1, τ_2, ..., τ_s with $p = 20 \cdot s$, where τ_1 releases J_1, J_{1+s}, $J_{1+2\cdot s}$, $\cdots$, and τ_2 releases J_2, J_{2+s}, $J_{2+2\cdot s}$, $\cdots$ and so on. These different tasks can be scheduled as exclusive tasks on isolated processors or in separate CBS instances. For this example, the task is divided into two tasks ($s = 2$), each task scheduled in a CBS with maximum budget $Q = 15$ and period $T = 20$. An illustration is provided in Fig. 11.3. On average, a job requires 21.8 units of computation, and the server provides 30 units for each job arrival. Due to the varying computation times, some jobs will not start at their arrival time. For instance, if a job J_j with computation time 38 starts at its arrival time, the next job in the same server, J_{j+2} will begin the latest 13 time units after its arrival, at $a_{j+2} + 13$. This job will be waiting even if other servers are idle at this time. If the computation time of J_{j+2} is also 38, it may finish at $a_{j+2} + 66$, and miss its deadline as illustrated in the separate queues case of Fig. 11.3. From simulation with the reservation of competing servers positioned at the beginning of each period, we see that the deadline miss rate is approximately 1%. A server is idle and not throttled for about 21% of the time.

Using a joint job queue and starting a waiting job as soon as any server is available reduces the risk of missing a deadline. Simulating the same task served by the same CBS instances sharing a joint job queue, the proportion of time where at least one server is idle and not throttled is about 41%, and the deadline miss rate reduces to approximately 0.19%. J_{j+2} is delayed as a direct effect of the long job J_j, but if J_{j+1} has a short computation time, J_{j+2} starts latest at $a_{j+2} + 10$. J_{j+2} will still meet its deadline if it has long computation time as illustrated in the joint queue case of Fig. 11.3. Using a joint queue for time-varying jobs, such as an MPC-task, decreases the deadline miss rate.

Example 11.4.2. *We use the task in Example 11.4.1, but with a different computation time probability mass function where long computation times are more common. In this example,* $(\mathbb{P}[\mathcal{C} = 20] = 0.4, \mathbb{P}[\mathcal{C} = 38] = 0.6)$.

The task in Example 11.4.2 has an average computation demand of 30.8 units per job, higher than the computational resource of 30 that is provided by

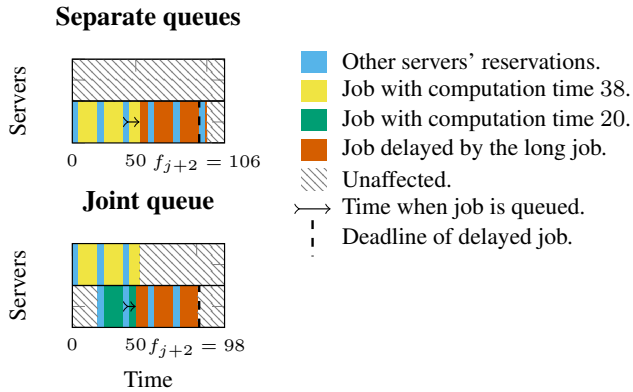


Fig. 11.3: Illustrations of jobs with delayed start due to a long computation time in Example 1. Arrows indicate the time with a non-empty queue.

the servers. This means that the queues will soon start growing, and all jobs will miss their deadlines.

Although job dismissal often is not explicitly implemented in schedulers, it may be implicit. Consider a single-thread task scheduled with `SCHED_DEADLINE` in Linux. If a job overruns and the task is implemented to self-suspend until the time for the next job activation, it will immediately continue with the next job. In a sequential task where jobs retrieve the most up-to-date data, this implicitly implements the skip-next policy [41] if the next job starts once new data has arrived. However, with the need for parallel jobs comes a need to specify the data connected to each job. The most straightforward way to handle this is to put the data or jobs in a queue, leading to the risk of unbounded queue length and tardiness in an overload situation. Support for dismissal of items from the queue is a remedy to this problem.

Problem Formulation

Devise a job acceptance policy with low overhead for JAMS, with the following properties when applied to a task as outlined in Section 11.3:

1. Out of jobs that are accepted, a configured proportion ϕ meet their deadline.

2. If the JAMS is in a long overload interval, the proportion of dismissed work approaches the proportion of computation requirement that exceeds the computation resource provided by the servers.

3. Given computational resource that exceeds the average requirement, a bound on the worst case job dismissal probability is derived, based on the arriving work in each possible interval.

11.5 JAMS and Job Acceptance Test

In this section, we outline the operation of the proposed JAMS, and provide an analysis of its properties.

11.5.1 JAMS Operation

Jobs arrive at the JAMS, which can access the states of its CBS. The JAMS is configured with the task relative deadline D, a quantile ϕ of jobs that shall meet their deadline if they are not dismissed, an estimate of this computation time quantile c_ϕ, and the task's WCET $c^\uparrow$. To ensure that resources are provided to jobs with a reasonable chance of meeting their deadline, a job J_j is transferred to a server S_i only if the server can guarantee sufficient resources prior to the deadline of J_j. The runtime operation of a JAMS and one of its CBS servers is illustrated in Fig. 11.4.

When a job arrives to the JAMS, and there exist CBS in state **Idle**, the arriving job is offered to these CBS in arbitrary order. If there is no such CBS that accepts the offer, the job is added to the FIFO job queue with its arrival time. A CBS with more suitable q and d may pull the job from the queue later. When a job is pushed to the queue, any jobs at the front of the queue that cannot be provided sufficient computational resources prior to their deadline by any server are dismissed. Then the arriving job (with its arrival time) is added to the job queue.

When a CBS S_i is offered an arriving job or tries to find a job on the queue to pull, it considers the amount of computation time it can guarantee prior to the job's deadline. For a job J_j with deadline $a_j + D$, the guaranteed computation

time till the job deadline is denoted as $g_{i,j}$. A job is accepted if at least c_ϕ computation time can be guaranteed prior to the deadline, i.e.:

$$g_{i,j} \geq c_\phi. \tag{11.2}$$

When a server attempts to pull a job from the queue, it first considers the job at the front of the queue. If this is not accepted, it continues with the next job until a job is accepted or the end of the queue is reached.

Race conditions may occur that are not included in the illustration Fig. 11.4. We note that multiple servers may simultaneously access the job queue. Two versions of the JAMS are considered and evaluated. One uses the computation time quantile configured from the start. The other estimates the quantile from the computation times of completed jobs. In this case, c_ϕ of the JAMS is updated according to the P^2 algorithm [28] when jobs are completed in the servers. The quantile estimate will be the computation time value where the proportion ϕ of observed computation times are below c_ϕ.

Example 11.5.1. *Let us consider Example 11.4.1, scheduled by a JAMS with* $Q = 15$, $T = 20$ *and* $n = 2$ *as illustrated in Fig. 11.3. Let* $c^\uparrow = 40$ *and require* $\phi = 0.95$ *of started jobs to meet their deadlines.*

For this example, the computation time quantile is $c_\phi = 38$. If the guaranteed computation time prior to a job's deadline is at least 38 it will be accepted, otherwise it will be left on the queue.

Example 11.5.2. *Let us again consider Example 11.4.1, scheduled by a JAMS with* $Q = 15$, $T = 20$ *and* $n = 2$ *as illustrated in Fig. 11.3. Let* $c^\uparrow = 40$ *and require* $\phi = 0.8$ *of started jobs to meet their deadlines.*

For this example, the computation time quantile is $c_\phi = 20$. If the guaranteed computation time prior to a job's deadline is at least 20, it will be started. Otherwise, it will be left in the queue.

11.5.2 Maximum Job Queue Length

Since jobs at the front of the queue that cannot be provided sufficient computational resources prior to their deadline by any server are dismissed when a new

job arrives, the maximum number of queued jobs can never exceed the number of jobs arriving during the task's relative deadline D. With the maximum number of jobs arriving instantaneously κ, and the minimum separation time of subsequent job arrival instants p, this means that the maximum job queue length is bounded by $\kappa \cdot \left\lceil \frac{D}{p} \right\rceil$.

11.5.3 Job Queue Waiting Time

We want to bound the waiting time of a job J^* that arrives at the JAMS. Denote the remaining work of the at most n jobs executing in the servers at the arrival of J^* with the random variable $\mathcal{B}$, and the work of all queued jobs at the arrival with the random variable $\mathcal{L}$. The waiting time of J^* is bounded in Theorem 11.5.1.

Theorem 11.5.1. *The waiting time $\mathcal{W}$ of a job J^* that arrives at a JAMS where $\mathcal{B}$ is the remaining work of currently running jobs, and $\mathcal{L}$ is the total work of jobs on the queue at the arrival of J^* is bounded by Eq.* (11.3).

$$\mathcal{W} \leq T \cdot \left\lceil \frac{\mathcal{B} + \mathcal{L}}{n \cdot Q} \right\rceil \tag{11.3}$$

Proof. The mean work to complete in a server before starting J^* is $\frac{\mathcal{B}+\mathcal{L}}{n}$, and this is completed latest at $T \cdot \left\lceil \frac{\mathcal{B}+\mathcal{L}}{n \cdot Q} \right\rceil$. At the latest at this point at least one server is available to start executing the work of J^*. □

11.5.4 Guaranteed Computation Time

When a server S_i requests to pull a job from the queue at time t, the guaranteed computation time $g_{i,j}$ provided by S_i prior to the deadline of J_j at the front of the queue is calculated. To simplify these expressions we denote the difference between J_j's deadline and the server's current deadline by $\delta_{i,j} = a_j + D - d_i$. If $\delta_{i,j} < 0$, then a part of the leftover budget q_i of the current server activation can be guaranteed. The server's execution ends latest at $t + U_{i\Sigma} \cdot (d_i - t)$, considering the total utilization $U_{i\Sigma}$ on the processor. If $\delta_{i,j} \geq 0$, the full leftover budget q_i of the current activation is guaranteed, plus Q units for each full server period

after d_i before J_j's deadline, and possibly a part of the budget in the last server period before the job's deadline. The guaranteed computation time $g_{i,j}$ can be computed as:

$$g_{i,j} = \begin{cases} \left[q_i - \left[U_{i\Sigma} \cdot (d_i - t) - (a_j + D - t)\right]^+\right]^+, & \delta_{i,j} < 0, \\ q_i + Q \cdot \left\lfloor \frac{\delta_{i,j}}{T} \right\rfloor + & \\ \quad + \left[Q - \left[U_{i\Sigma} \cdot T - \delta_{i,j} \mod T\right]^+\right]^+, & \delta_{i,j} \geq 0, \end{cases} \tag{11.4}$$

where $[x]^+ := \max(x, 0)$.

For analysis purposes, we bound $\mathcal{G}$ in Theorem 11.5.2.

Theorem 11.5.2. *A job J^* that arrives to a JAMS with remaining work on the servers $\mathcal{B}$ and queued work $\mathcal{L}$ will be guaranteed at least $\mathcal{G}$ computation resource prior to its deadline as outlined in Eq.* (11.5)

$$\mathcal{G} \geq Q \cdot \left\lfloor \frac{D}{T} \right\rfloor - Q \cdot \left\lceil \frac{\mathcal{B} + \mathcal{L}}{n \cdot Q} \right\rceil \tag{11.5}$$

Proof. The waiting time $\mathcal{W}$ of J^* is bounded in Eq. (11.3). The time remaining until the deadline when the job is pulled by a CBS is $D - \mathcal{W}$. The guaranteed computation time of a server within this time is at least Q times the number of full server periods in $D - \mathcal{W}$, giving:

$$\begin{aligned} \mathcal{G} \geq Q \cdot \left\lfloor \frac{D - \mathcal{W}}{T} \right\rfloor &\geq Q \cdot \left\lfloor \frac{D}{T} \right\rfloor - Q \cdot \left\lfloor \frac{T \cdot \left\lceil \frac{\mathcal{B}+\mathcal{L}}{n \cdot Q} \right\rceil}{T} \right\rfloor \\ &= Q \cdot \left\lfloor \frac{D}{T} \right\rfloor - Q \cdot \left\lceil \frac{\mathcal{B} + \mathcal{L}}{n \cdot Q} \right\rceil \end{aligned}$$

□

11.5.5 Dismissal Probability

A job J_j will be dismissed if $g_{i,j} < c_\phi, \forall i$. Therefore the probability of dismissing a job is bounded in Eq. (11.6), by inserting the bound on the guaranteed computation time in Eq. (11.5), and in the last step using the bound on the

served work $\mathcal{B} \leq n \cdot c^{\uparrow}$.

$$\begin{aligned}
\mathbb{P}\left[\mathcal{G} < c_\phi\right] &\leq \mathbb{P}\left[Q \cdot \left\lfloor \frac{D}{T} \right\rfloor - Q \cdot \left\lceil \frac{\mathcal{B}+\mathcal{L}}{n \cdot Q} \right\rceil < c_\phi \right] \\
&= \mathbb{P}\left[\left\lceil \frac{\mathcal{B}+\mathcal{L}}{n \cdot Q} \right\rceil > \left\lfloor \frac{D}{T} \right\rfloor - \frac{c_\phi}{Q} \right] \\
&\leq \mathbb{P}\left[\left\lceil \frac{\mathcal{L}}{n \cdot Q} \right\rceil > \left\lfloor \frac{D}{T} \right\rfloor - \frac{c_\phi}{Q} - \left\lceil \frac{c^{\uparrow}}{Q} \right\rceil \right]
\end{aligned} \tag{11.6}$$

Dismissals in an Overload Scenario

Consider a scenario where the total work on the queue remains high for a sufficient amount of time so that all servers are running at full capacity, and some jobs are being dismissed because of insufficient guaranteed computation time prior to their deadline. At this point, the average work completion rate by the servers is $n \cdot \frac{Q}{T}$. Let the average work arrival rate during the overload scenario be $\gamma \cdot n \cdot \frac{Q}{T}, \gamma > 1$. This implies that the proportion of dismissed work is $1 - \gamma^{-1}$, consistent with steady-state queuing theory analysis of an overload system with abandonment [50]. The dismissal decision for a job is independent of the computation time of the specific job. If computation times are independent random variables, the proportion of dismissed jobs is also $1 - \gamma^{-1}$. The dismissal probability of a specific job may be higher or lower due to variations in the work arrival rate and is bounded by Eq. (11.6).

We consider Example 11.4.2. In this case, computation times are i.i.d. and the required average computational requirement is 30.8 per job arrival, and the provided resource is 30. This gives $\gamma = \frac{30.8}{30} \approx 1.027$ and a dismissal rate of about 2.6%.

Dismissals With Sufficient Average Capacity

For a system with sufficient average capacity, a bound on the dismissal probability of a job in JAMS is outlined in Theorem 11.5.3.

Theorem 11.5.3. *For a JAMS that is not overloaded in any intervals longer than Δ_L, the probability p_d that a job is dismissed is bounded by Eq.* (11.7)*.*

$$p_d \leq \max_{\Delta} \left(\mathbb{P} \left[\left\lceil \frac{\Upsilon(\Delta) + \mathcal{B}}{n \cdot Q} \right\rceil > \frac{\Delta + D}{T} - \left\lceil \frac{c_\phi}{Q} \right\rceil \right] \right) \tag{11.7}$$

Proof. A system that is not overloaded in any long interval has some points in time when the queue is empty, and at least one server is ready to start executing an arriving job immediately. Denote a time when the last remaining idle server goes to active state with t_0. Now we consider a job J^* arriving at $t_0+\Delta, \Delta > 0$, assuming that all servers are busy in the interval $[t_0, t_0 + \Delta]$. J^* will start at the latest when one server is no longer processing the remaining work of at most $n - 1$ jobs that were running at t_0 or work that arrived in the interval except for the work of J^*. If J^* starts before $t_0 + \Delta + D - T \cdot \left\lceil \frac{c_\phi}{Q} \right\rceil$, it will not be dismissed.

Denote the remaining work of jobs running at t_0 as $\mathcal{B}$ and the work arriving in this interval except for the work of J^* as $\Upsilon(\Delta)$. J^* will not be dismissed if:

$$t_0 + T \cdot \left\lceil \frac{\Upsilon(\Delta) + \mathcal{B}}{n \cdot Q} \right\rceil \leq t_0 + \Delta + D - T \cdot \left\lceil \frac{c_\phi}{Q} \right\rceil \tag{11.8}$$

Restructuring of this condition gives an upper bound on the probability that J^* is dismissed as:

$$\mathbb{P} \left[\left\lceil \frac{\Upsilon(\Delta) + \mathcal{B}}{n \cdot Q} \right\rceil > \frac{\Delta + D}{T} - \left\lceil \frac{c_\phi}{Q} \right\rceil \right] \tag{11.9}$$

For an arbitrary job J_j, if it arrives to a JAMS with at least one server idle, it will not be dismissed due to Assumption 11.3.2, and the bound is trivial. If it arrives to a JAMS with all servers active, there exists a Δ to the most recent point when the last server went to active state. Then Eq. (11.9) bounds the dismissal probability of J_j with this Δ. Eq. (11.7) is greater than or equal to Eq. (11.9), so it bounds the dismissal probability of J_j. □

A task with average computation time requirement below that provided by the servers will experience no dismissals if for each interval length Δ, the arriving work $\Upsilon(\Delta)$ and the work $\mathcal{B}$ remaining to be served at the start of the

interval when the last server goes to active are bounded by Eq. (11.8). For a system where $D > T \cdot \left(\left\lceil \frac{c^{\uparrow}}{Q} \right\rceil + \left\lceil \frac{c_{\phi}}{Q} \right\rceil \right)$ and work arrival rate bound $\rho \leq \frac{n \cdot Q}{T}$ we have dismissal probability at most ϵ in Eq. (11.9) for all $\Delta \geq \Delta_B$ from Assumption 11.3.1.

As Δ grows, Eq. (11.8) goes toward the condition that the provided computational resource needs to be higher than the average demand. $\Upsilon(\Delta) + \mathcal{B}$ in Eqs. (11.7) to (11.9) is a sum of computation time random variables. If computation times are independent random variables, the sum can be calculated by convolution. If computation times are correlated, convolution of upper bounding probabilistic WCET (pWCET) distributions can be applied [19], or a bound can be derived in other ways [36, 17]. If computation times are described by Markov Models, these can be utilized to calculate the sum [23, 24].

Let us go back to Example 11.5.1. We need to consider Δ as multiples of 20, as jobs are released with $p = 20$. For $\Delta = 20$, we have a bound on the dismissal probability as $\mathbb{P}\left[\left\lceil \frac{\Upsilon(20)+\mathcal{B}}{2 \cdot 15} \right\rceil > \frac{20+60}{20} - \left\lceil \frac{38}{15} \right\rceil\right] = \mathbb{P}\left[\left\lceil \frac{\Upsilon(20)+\mathcal{B}}{30} \right\rceil > 1\right]$. In this case $\mathcal{B}$ may contain almost the full work of 1 job, if the queue was empty just a short time prior to t_0. $\Upsilon(20)$ contains 1 job, that arrived at t_0. It is clear that with this task and configuration, we cannot guarantee that a job arriving one period after the start of a queue is not dismissed. With computation times 20 or 38, two jobs will not fit in the total budget of 30. For $\Delta = 40$ and $D = 60$, $\Upsilon(40)$ contains two jobs and the dismissal probability bound is $\mathbb{P}\left[\left\lceil \frac{\Upsilon(40)+\mathcal{B}}{30} \right\rceil > 2\right]$. All three jobs in $\mathcal{B} + \Upsilon(40)$ need to have computation time 20 to ensure no dismissal, and the probability of this is 0.9^3. The dismissal probability bound for jobs arriving within two periods from t_0 is $1 - 0.9^3 = 0.271$. For $\Delta = 60$, we need to consider a total of 4 jobs in $\mathcal{B} + \Upsilon(60)$. For $D = 60$, we have $\mathbb{P}\left[\left\lceil \frac{\Upsilon(60)+\mathcal{B}}{30} \right\rceil > 3\right]$, and all four jobs need to have computation time 20 to ensure no dismissal, the probability of this is about 0.34.

Considering the lower computation time quantile in Example 11.5.2, this will lead to lower dismissal probability bounds, as the term $\left\lceil \frac{38}{15} \right\rceil$ is replaced by $\left\lceil \frac{20}{15} \right\rceil$.

Example 11.5.3. *Let us consider Example 11.5.1, but now with a longer rela-*

tive deadline of 100.

With the relative deadline of 100, we would instead have the dismissal probability bound for $\Delta = 20$ as $\mathbb{P}\left[\left\lceil \frac{\Upsilon(20)+\mathcal{B}}{2\cdot 15} \right\rceil > \frac{20+100}{20} - \left\lceil \frac{38}{15} \right\rceil\right] = \mathbb{P}\left[\left\lceil \frac{\Upsilon(20)+\mathcal{B}}{30} \right\rceil > 3\right]$. We see that even in the worst case where both jobs contributing work to $\Upsilon(20) + \mathcal{B}$ have computation time 38, we can guarantee that there is no dismissal at this point.

With $\Delta = 40$, we have $\mathbb{P}\left[\left\lceil \frac{\Upsilon(40)+\mathcal{B}}{30} \right\rceil > 4\right]$, and the dismissal probability is 0.

With $\Delta = 60$, $\mathbb{P}\left[\left\lceil \frac{\Upsilon(60)+\mathcal{B}}{30} \right\rceil > 5\right]$ at least one of the jobs needs to be short to avoid a dismissal, so the dismissal probability bound equals the probability of all jobs being long, 0.1^4.

11.5.6 Probability of Meeting the Deadline

We show that for a job that is accepted by a server, the deadline will be met with at least probability ϕ, provided the probability is at least ϕ that the job's computation times is below c_ϕ. For independent computation times, ϕ represents a bound on the probability that $\mathcal{C} < c_\phi$ for each job that is run. For correlated computation times, individual jobs may have higher or lower probability of exceeding the quantile. In this case the average ratio of started jobs that meet their deadline to started jobs is at least ϕ.

Theorem 11.5.4. *Assuming i.i.d. computation times, a job J_j that is accepted by a CBS in JAMS has at least probability ϕ of meeting its deadline.*

Proof. A job that is accepted is guaranteed at least c_ϕ computation time prior to its deadline. From this if follows that $\mathbb{P}[\mathcal{R} \leq D] \geq \mathbb{P}[\mathcal{C} \leq c_\phi] \geq \phi$. □

Theorem 11.5.5. *The rate of jobs accepted by a CBS in JAMS that meet their deadline is at least ϕ.*

Proof. A job J_j that is accepted is guaranteed at least c_ϕ computation time prior to its deadline. The outcome of the acceptance test is independent of the computation time of J_j. Therefore, the rate of accepted jobs that meet their

deadlines is at least the rate of jobs with computation time below c_ϕ, that is ϕ. □

We note that using a precomputed computation time quantile derived from a pWCET distribution that upper bounds the computation time distribution ensures that the probability bound on meeting the deadline holds for each accepted job even with correlation.

A general bound on the probability of meeting the deadline, that holds even for jobs with computation time $c_j > c_\phi$ is derived in Eq. (11.10).

$$\begin{aligned}\mathbb{P}\left[\mathcal{C} \leq \mathcal{G}\right] \geq & \mathbb{P}\left[\mathcal{C} \leq Q \cdot \left\lfloor \frac{D}{T} \right\rfloor - Q \cdot \left\lceil \frac{\mathcal{B}+\mathcal{L}}{n \cdot Q} \right\rceil \right] \\ = & \mathbb{P}\left[\left\lceil \frac{\mathcal{B}+\mathcal{L}}{n \cdot Q} \right\rceil + \frac{\mathcal{C}}{Q} \leq \left\lfloor \frac{D}{T} \right\rfloor \right]\end{aligned} \tag{11.10}$$

We return to the comparison of Examples 11.5.1 and 11.5.2. We have seen that a lower computation time quantile leads to lower dismissal probability, but this comes at the price of a higher probability of deadline miss for started jobs.

Finally, we note that an upper bound on the dismissal probability p_d, and a lower bound on the probability that an accepted job meets its deadline ϕ, imply a lower bound on the probability of meeting the deadline for every job as $(1 - p_d) \cdot \phi$.

11.6 Implementation and Overheads

The mechanism described in Section 11.5 has been implemented as a multi-threaded C program on the Linux Operating System, managing a limited-size shared queue that exposes special blocking operations to push jobs into the queue and pull admitted jobs out of it, alongside performing the actual dismissal operations described in Section 11.5. Our JAMS implementation may optionally use a kernel module we realized for faster and more accurate access to the `SCHED_DEADLINE` state parameters at runtime, as described below.

11.6.1 JAMS Push and Pull Operations

The push operation is a simple blocking operation that pushes jobs into a FIFO-ordered queue, managed through a classical circular buffer. In the JAMS design, jobs are pushed all with the same relative deadline; thus, their submission order corresponds to a deadline-based order. In the experimentation described below, we configured the queue size never to hit the size limit in a push operation. After enqueuing a job for processing, the push operation notifies other threads possibly waiting on a pull through a condition variable. The pull operation is more involved and contains the majority of our JAMS implementation: we retrieve the runtime left and absolute deadline of the CBS server (see below), then we scan the jobs waiting in the queue, starting from the earliest submitted job, checking whether the budget available till the deadline is sufficient for the estimated percentile of the job computation time, using Eq. (11.2): in such a case, the job is pulled out of the queue and returned to the caller for processing, otherwise we move forward to check the next job in the queue. In the latter case, the job is not immediately dismissed but left for the other threads in the JAMS CBS group, which will, in turn, evaluate the job based on their own CBS instantaneous parameters. A job that is not picked up by any server is eventually dismissed. Whenever the pull operation cannot find admissible jobs to process in the queue, it blocks on a condition variable, waiting for a push operation.

11.6.2 Reading `SCHED_DEADLINE` Parameters

The `SCHED_DEADLINE` scheduler available on Linux has a direct API to read the statically configured maximum runtime, relative deadline, and period of a CBS server (through the `sched_getattr()` syscall), but it lacks an API to retrieve the leftover runtime and absolute deadline. However, these parameters can be accessed through the `/proc` filesystem, reading the per-task `sched` special file, where the `dl.runtime` and `dl.deadline` parameters can be found for tasks under the `SCHED_DEADLINE` policy. The runtime value accessible this way is only guaranteed to be updated at each periodic system tick, with a frequency of HZ. Reconfiguring the kernel with a 1ms HZ results in a more accurate reading. However, the `/proc` interface is designed for debug-

ging rather than for production use, so it suffers from inherent inefficiency. For example, all quantities are wastefully formatted in decimal notation from the kernel space and have to be converted back into user space. Therefore, we realized a kernel module that allows for accurately reading the `SCHED_DEADLINE` parameters directly in binary format, with much greater efficiency. On the Raspberry Pi 4 board used for our experimentation, reading the current parameters using the `/proc` interface resulted in a $228 \pm 35\mu s$ per-reading overhead, while with our kernel module, this was reduced to a $10.8 \pm 2.4\mu s$ per-reading overhead. Note that, in JAMS, worker threads need to retrieve this information each time a job is evaluated for processing to apply the acceptance policy properly.

11.6.3 JAMS Overheads

This subsection discusses the computational and memory overheads due to using JAMS, in its current implementation. The time required for JAMS push and pull operations is bounded by a constant time per call. The most significant overhead is observed during 'pull' when reading the server parameters, for which a significant improvement is discussed in Section 11.6.2 (reducing it from $228\mu s$ down to $10.8\mu s$). Regarding how often we pull, in a case where JAMS is mainly idle, nearly every server will wake up and attempt a pull at each job arrival. When JAMS servers are mainly busy, each server will perform a pull per processed job, roughly with a rate equal to the arrival rate divided by the number of servers. With our implementation and evaluation set-up, the average pull time with the improvement discussed in Section 11.6.2 is $49\mu s$, and the average push time is $9\mu s$. In our evaluation scenarios, job computation times are $10-300ms$, making those overheads quite negligible from a practical standpoint. Regarding the memory overhead, arrival times must be stored for all queued jobs.

11.7 Experimental Evaluation

In this section, we provide results from the experimental evaluation[1] of our proposed JAMS mechanism, implemented as detailed in Section 11.6. The behavior of the proposed JAMS and dismissal mechanism is evaluated in the presence of both synthetic and realistic workload scenarios. We compare the deadline miss rate and job dismissal rate for different configurations and workloads. We also show how application of the JAMS acceptance policy affects the response times throughout an execution sequence. A comparison is performed between applying the JAMS dismissal policy with a statically precomputed quantile of the computation times versus using an online estimated quantile. Comparisons are performed against a baseline with multiple servers that always pull the first available job from the queue without applying any dismissal policy.

Two types of workload are considered in the evaluation: synthetically generated computation times with a lognormal distribution and recorded computation times from an MPC task.

11.7.1 Computation Time Data

The MPC task is based on the Unmanned Ground Vehicle (UGV) path planning with obstacles example of the libmpc++ library [42]. The Sequential Quadratic Programming algorithm SLSQP [30] from the NLopt library [29] is used for the optimization. Predictions and control are computed over a 6-step horizon. A list of waypoints is used, and when a waypoint is reached, the next is obtained from the list in a circular manner. The simulated UGV is run in two environments, with smaller or larger obstacles, resulting in lighter or heavier computational load. The starting point, waypoint list, and configuration of the optimization are the same. The libmpc++ optimization time logging is modified to use the `clock_gettime(CLOCK_THREAD_CPUTIME_ID)` syscall, and log the computation time of the optimization rather than the response time.

10 runs of 5000 optimization steps are performed in each environment. The computation time data collection is performed on a Raspberry Pi 3B+ with

[1] Code and data for the artifact evaluation are available at https://github.com/annafriebe/RTAS_25_JAMS_AE.

PREEMPT_RT where frequency scaling and USB have been disabled. The task is scheduled with the highest priority FIFO scheduling and pinned to a core with the cpuset utility. Computation times are retrieved from the libmpc++ log files. The experimental Cumulative Distribution Function (CDF) of the MPC computation times are reported in Fig. 11.5. There is a high degree of correlation among the computation times, since they are affected by the UGV state dynamics.

We also used synthetic computation times that have been generated with lognormal distributions. These have been shown to be a good fit for computation time distributions in some applications [44]. We have generated 10 lognormal traces, drawing the average randomly from the range $[40, 60]$ms and the standard deviation from the range $[30, 40]$ms. The distribution is bounded to the range $[10, 160]$ by discarding samples outside this range. The synthetically generated computation time CDFs are reported in Fig. 11.6. Means and standard deviations shown in the figure are empirical from the samples in the bounded range, with the exception of the last dotted line. In this case these are the mean and standard deviation of the lognormal distribution prior to bounding the range.

11.7.2 Evaluation Program and Test-Bed Setup

The described JAMS mechanism has been used in the program we realized for the experimental validation, where one thread was dedicated to submitting jobs to JAMS via the push operation, while n worker threads used the pull operation. Each thread was attached to a SCHED_DEADLINE reservation with configurable dl_runtime and dl_deadline = dl_period parameters. Reservations were configured in partitioned EDF mode[2]. Job computation times were provided as trace files input to the program, produced according to the workload scenarios described in Section 11.7.1. The thread submitting jobs to the JAMS has been periodically activated, with a specified job inter-arrival period. The worker threads have been using the special pull operation to

[2]This was obtained via disabling the in-kernel access control by writing -1 to sched_rt_runtime_us in /proc/sys/kernel, then setting the needed affinity masks on SCHED_DEADLINE threads.

retrieve the admitted jobs and then process them by performing wasteful computations for the amount of time of each job, as instructed by the trace file that was read at the beginning of the program. Finally, they measured the response time for the job, storing it in an in-memory array. At the end of the program, all the stored response times have been dumped into an output file. In all experiments, jobs are released periodically every 80ms, equal to the server period T.

The experiments have been run on a Raspberry Pi 4 Model B board equipped with an Arm Cortex-A72 quad-core CPU and 3.7GiB of RAM, with the CPU frequency locked at the maximum value via `cpufreq`. The JAMS was configured with 2 worker threads pinned down on cores 2 and 3, and the thread submitting requests pinned down on core 1. The 3 cores have been isolated from the general OS workload using the `isolcpus` boot parameter of the kernel. Additionally, all experiments were carried out at runlevel 1 to avoid starting unnecessary services on the platform.

11.7.3 Experiment 1 – MPC Traces

In the first experiment, we performed a program run using the 10 traces from the MPC use-case whose distribution is reported in Fig. 11.5. We have reported the obtained deadline miss percentage and dismissed jobs percentage in Fig. 11.7. The relative task deadline is 480 ms, chosen from the MPC 6-step horizon and the 80 ms period. The top plot shows the results obtained processing the 10 more demanding traces, tagged "heavy", using a per-CBS allocation bandwidth of 60%. When using no dismissal, the system turns out to be quite overloaded throughout the run, resulting in deadline miss rates between 30% and 60%, as visible from the "No dismissal" violet dots cloud at the bottom right of the plot. When enabling the JAMS acceptance policy configured statically with the known 95th percentile of the input traces, we obtain roughly 1% of deadline misses at the expense of a 10% of job dismissals (green orthogonal crosses cloud). The results are aligned with our deadline-miss theoretical expectations in Theorem 11.5.5, as the system would be supposed to guarantee at least a 95% of deadline hit rate under these conditions. Switching to using a dynamic percentile estimator, we obtain the green oblique crosses cloud, having a slightly lower dismissal rate at the expense of doubling the deadline miss

ratio, which stays safely within the 5% design bound (highlighted as the green vertical dashed bar in the plot).

Moving to the plot's blue and brown dots cloud series, we can see a similar behavior obtained by configuring JAMS with a different target percentile, namely 90th and 85th (blue and brown series, respectively). In both cases, the obtained results confirm what just discussed above, with the difference that, when JAMS uses a lower computation time percentile in its configuration, it tends to admit a higher number of jobs, obtaining a higher percentage of deadline misses, which keeps staying safely below the theoretical bounds (10% and 15% marked with a blue and brown dashed vertical bar in the two plots).

To mitigate the need to dismiss jobs with a heavy workload, we can assign more computational resources to JAMS, increasing the per-CPU bandwidth from 60% to 70%. The effects of such a change on the various configurations are shown in the second plot of Fig. 11.7. In this case, the baseline runs exhibit significantly fewer deadline misses (reduced from the previous 30%-60% down to 15%-30%); however, they are still above the desirable threshold of 5%, for example. The JAMS mechanism, in this case, also proves to be beneficial, managing to keep the deadline-miss rate for the processed jobs within the design bounds (the usual three percentiles are shown in the picture), applying a dismissal rate roughly equal to half the one of the previous case with the per-CPU bandwidth of 60%.

The bottom plot in Fig. 11.7 shows the results obtained processing the 10 lighter traces, tagged "light", using a per-CBS allocated bandwidth of 40%. Under these conditions, our baseline without dismissals results in a deadline-miss rate between 5% and 16%, visible in the "No dismissals" violet dots at the bottom right of the plot. Enabling our JAMS dismissal policy, with a configured 95% of target deadline-miss bound, we obtain deadline-miss rates between 0.5% and 1.5% (again safely below the design bound), at the expense of a dismissal rate between 1.5% and 3.5%.

In the collected experimental data, a final piece of information worth a look at is the number of consecutive jobs that miss their deadline or are dismissed. Fig. 11.8 reports the response times obtained for the first 1000 jobs in the experiment using the MPC light trace 0, without any dismissal policy (blue dots), versus using JAMS with a statically configured 95% percentile (red dots), where

job dismissals are highlighted by red crosses. It is clear that without a dismissal policy, transient overloads lead to long periods with no jobs producing timely results.

11.7.4 Experiment 2 - MPC Traces with Real-Time Load

We also experimented with the JAMS mechanism running alongside other real-time workload on the platform. As JAMS is designed around using CBS and partitioned EDF for temporal isolation among multiple real-time tasks, the additional load on the platform was also run through CBS reservations. More specifically, we performed a run of the heavy traces described above, served by 2 CBS servers occupying 60% of the CPUs 2 and 3, deploying on each of these 2 CPUs also additional 3 real-time tasks, attached to CBS reservations with runtime and period=deadline of (3ms,60ms), (5ms,100ms) and (7ms,140ms), for a total of additional 15% of real-time CPU workload.

JAMS was made aware of the total utilization of the CPUs where its CBS servers were deployed, so to correctly compute the available budget to deadline $g_{i,j}$ in Eq. (11.4). The obtained results are summarized in Fig. 11.9, where, alongside the usual "No dismissal" points visible at the bottom right of the plot, we can see the results from JAMS configured with 95th, 90th and 85th percentiles of the computation times distributions (in green, blue and brown dots, respectively). Results are reported with statically configured percentiles (orthogonal crosses) and dynamically estimated ones (oblique crosses). The expected percentile bounds on the deadline misses are represented with vertical dashed lines with the same color as the points corresponding to that configured percentile (green, blue, and brown for 95th, 90th, and 85th, respectively).

11.7.5 Experiment 3 - Lognormal I.I.D. Traces

We performed another experiment using the lognormal traces shown in Fig. 11.6. In this case, we used JAMS configured with 2 threads, 30% of per-CBS bandwidth, a relative task deadline of 480 ms, and the usual three percentile configurations. The obtained results are shown in Fig. 11.10. In this case, the lognormal traces were generated using different parameters drawn at random, as described earlier. Therefore, with the configured CBS bandwidth

assignment, we obtained a wide range of different results, evident from the "No dismissal" violet dots scattered throughout the X axis, from 2.5% to 100% of deadline miss rates. Interestingly, applying the JAMS mechanism to these cases, we consistently obtain a deadline-miss ratio below the configured target percentile (95th, 90th, and 85th, highlighted as dashed vertical lines as usual), obtained at the cost of applying a dismissal rate that also varies greatly across the cases, from 2% to nearly 50% of dismissals.

We also performed a run designed to validate the dismissal probability bound in Theorem 11.5.3. This was done by running our experimentation using the last trace shown as a dashed curve in Fig. 11.6, with JAMS configured as usual with 2 CBS servers with a 60% bandwidth and a job deadline of 240ms. Over the run, we experienced 0.04% of job dismissals. The theoretical dismissal rate bound from Eq. (11.6) is derived, calculating $\mathcal{B}+\mathcal{L}$ as a convolution of an increasing number of bounded lognormal distributions for several Δ as multiples of 80ms. The dismissal probability bound is 1.2%, observed for the shortest Δ.

11.8 Conclusion and Future Work

This paper introduced the Job Acceptance Multi-Server (JAMS) mechanism as an extension of the Constant Bandwidth Server (CBS) for efficient resource allocation in real-time systems. By leveraging a shared job queue across multiple CBS instances, JAMS enables dynamic resource sharing among tasks with varying computational demands and deadlines. Its targeted job acceptance and dismissal strategy prioritizes jobs with a high likelihood of meeting their deadline, limiting the impact of job tardiness and overloading. Experimental evaluation with synthetic and realistic workloads showed that JAMS significantly reduces the deadline miss rate compared to a baseline system that lacks a dismissal policy. More specifically, JAMS consistently meets design-bound deadline-miss ratios while managing resource demands through adaptive dismissal rates, showing robust performance under several diverse workload conditions.

In the future, we plan to extend the present work in various directions. On the analysis side, the theoretical bounds on dismissal probability presented

in the paper may be pessimistic in certain scenarios. Refining these bounds to achieve tighter estimates would improve the applicability of the approach in a broader set of scenarios. From an experimental perspective, more extensive experimentation is needed to assess possible scalability limitations of the proposed technique in the presence of several CBS servers pulling from the same shared queue. Albeit JAMS is proposed in the context of real-time embedded platforms, where the number of available CPUs might be limited, future embedded platforms seem to trend into featuring dozens of cores easily. Therefore, further testing on diverse architectures, including distributed and cloud/edge-based real-time systems, would provide insights into JAMS's scalability and resilience in a wide range of use-cases, such as real-time cloud computing [25, 53, 8, 46]. Finally, on the implementation side, our current JAMS prototype can certainly be improved by moving the queue management logic into kernel space. This would have the advantage of being able to easily access the CBS scheduling parameters of the `SCHED_DEADLINE` threads participating in a JAMS queue, with the ability to wake them up only when jobs can certainly be accepted.

Table 11.1: Overview of notation.

Symbol	Description
τ, J_j	Task, job j of τ.
κ	Maximum number of concurrently released jobs of τ.
p	Minimum separation of job release instants of τ.
D	Relative deadline of τ.
a_j, f_j	Arrival and finishing time of J_j.
c_j, $c^{\uparrow}$	Computation time of J_j and the WCET of τ.
$\mathcal{C}$, $\mathcal{R}$	Computation time and response time random variables.
c_ϕ^ω	The ϕ quantile of the computation times of τ for a given ω.
n	Number of servers.
S_i	Server i.
T_i, Q_i, U_i	Period, maximum budget and utilization of S_i.
$U_{i\Sigma}$	Total utilization on S_i's processor.
q_i, d_i	Remaining budget and deadline of S_i.
t, δt	Time, interval when a server is executed.
$g_{i,j}$	Guaranteed computation time from S_i prior to J_j's deadline.
$\delta_{i,j}$	Difference between the deadlines of J_j and S_i.
$\mathcal{W}$	Random variable, queue wait time.
$\mathcal{B}$	Random variable, remaining work of served jobs.
$\mathcal{L}$	Random variable, work of queued jobs.
$\mathcal{G}$	Random variable, guaranteed computation time of a job.
Δ	Time interval length.
$\Upsilon(\Delta)$	Random variable, work arriving in an interval of length Δ.
ρ	Bound on the work arrival rate.
p_d	Dismissal probability.

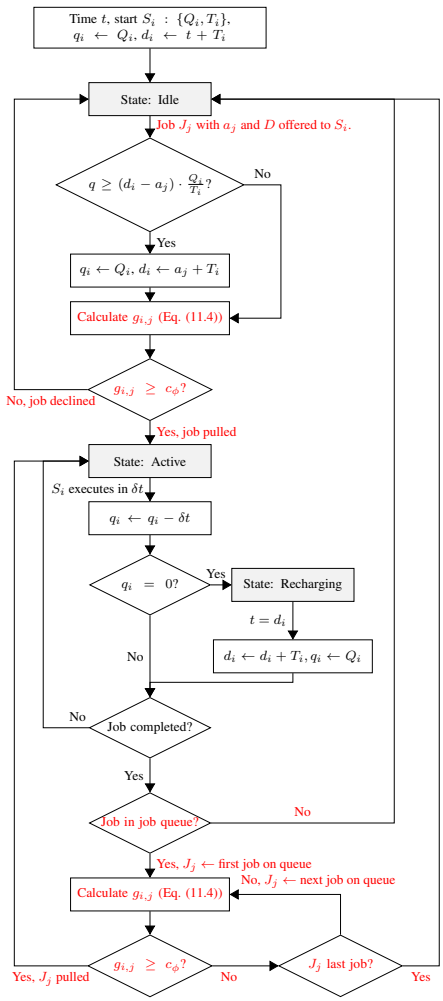


Fig. 11.4: Flowchart of a JAMS CBS.

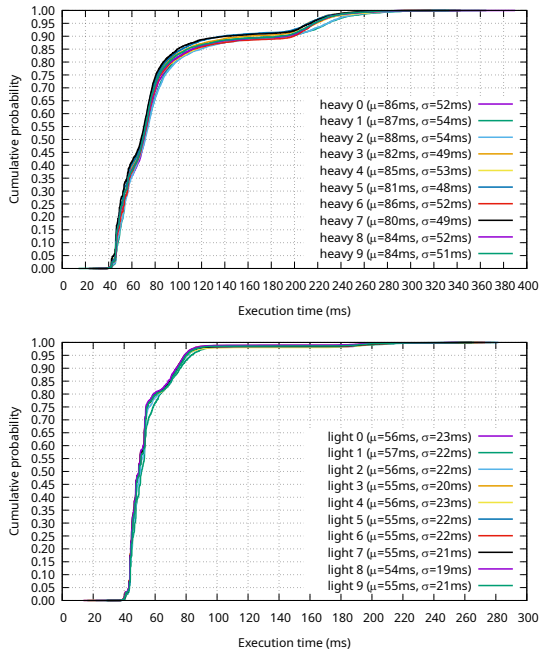


Fig. 11.5: Experimental CDFs of computation times for the heavier (top) and lighter (bottom) MPC traces.

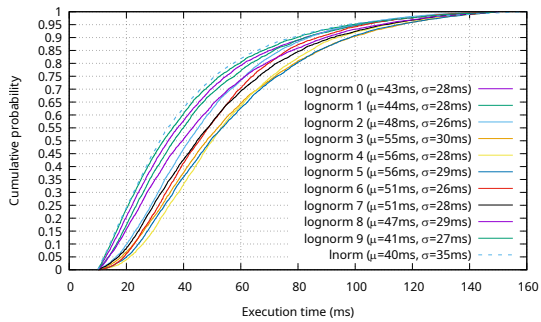


Fig. 11.6: Experimental CDFs of synthetic i.i.d. computation times generated from lognormal distributions (applying a truncation of the distributions in the [10,160]ms range).

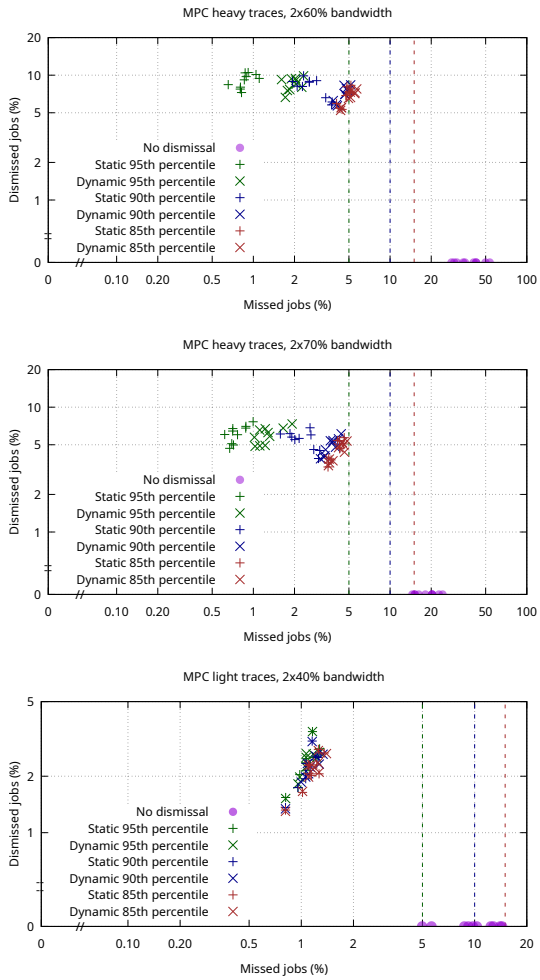


Fig. 11.7: Missed jobs (on the X axis) in % over the processed jobs, compared to the dismissed jobs % (on the Y axis), obtained with no dismissals baseline (violet dots), and JAMS with configurations of the percentile (various point colors), both when statically configured and dynamically estimated (orthogonal vs oblique crosses). Note that the X and Y axes are on a logarithmic scale, broken close to the origin so that 0 values can also be visualized.

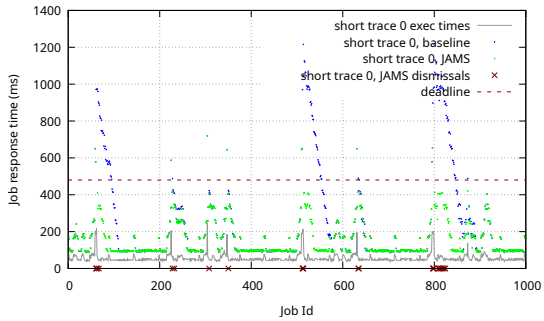


Fig. 11.8: Excerpt of computation times for the MPC lightweight trace 0 (gray line), and the per-job response times obtained using JAMS vs the baseline.

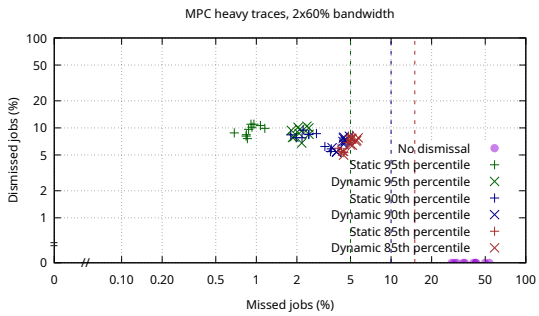


Fig. 11.9: Missed jobs (X axis) in % over the processed jobs, compared to the dismissed jobs % (Y axis). The board was hosting an additional 15% of per-CPU real-time reservations, spread across 3 real-time tasks on each CPU.

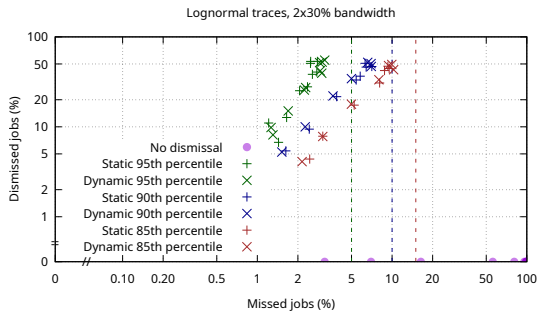


Fig. 11.10: Missed jobs (on the X axis) in % over the processed jobs, compared to the dismissed jobs % (on the Y axis).

Bibliography

[1] Luca Abeni, Alessio Balsini, and Tommaso Cucinotta. Container-based real-time scheduling in the linux kernel. *ACM SIGBED Review*, 16(3):33–38, 2019.

[2] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*, pages 4–13. IEEE, 1998.

[3] Luca Abeni and Giorgio Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27:123–167, 2004.

[4] Luca Abeni, Giuseppe Lipari, and Juri Lelli. Constant bandwidth server revisited. *Acm Sigbed Review*, 11(4):19–24, 2015.

[5] Mohamed Abouzahir, Abdelhafid Elouardi, Samir Bouaziz, Rachid Latif, and Abdelouahed Tajer. FastSLAM 2.0 running on a low-cost embedded architecture. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 1421–1426. IEEE, 2014.

[6] Shareef Ahmed and James H Anderson. Soft real-time gang scheduling. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 331–343. IEEE, 2023.

[7] Waqar Ali and Heechul Yun. RT-Gang: Real-time gang scheduling framework for safety-critical systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 143–155. IEEE, 2019.

[8] Remo Andreoli, Harald Gustafsson, Luca Abeni, Raquel Mini, and Tommaso Cucinotta. Optimal deployment of cloud-native applications with fault-tolerance and time-critical end-to-end constraints. In *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, UCC '23. ACM, December 2023.

[9] Sanjoy Baruah, Joël Goossens, and Giuseppe Lipari. Implementing constant-bandwidth servers upon multiprocessor platforms. In *Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 154–163. IEEE, 2002.

[10] Sanjoy K Baruah, Alan Burns, and Robert I Davis. Response-time analysis for mixed criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 34–43. IEEE, 2011.

[11] Iain Bate, Alan Burns, and Robert I Davis. An enhanced bailout protocol for mixed criticality embedded software. *IEEE Transactions on Software Engineering*, 43(4):298–320, 2016.

[12] Guillem Bernat, Alan Burns, and Albert Liamosi. Weakly hard real-time systems. *IEEE transactions on Computers*, 50(4):308–321, 2001.

[13] Alan Burns, Robert I Davis, Sanjoy Baruah, and Iain Bate. Robust mixed-criticality systems. *IEEE Transactions on Computers*, 67(10):1478–1491, 2018.

[14] Alan Burns and Robert Ian Davis. Mixed criticality systems-a review. Technical report, Department of Computer Science, University of York, February 2022.

[15] Giorgio C Buttazzo, John A Stankovic, et al. RED: Robust earliest deadline scheduling. In *Proc. of 3rd International Workshop on Resonsive Computing Systems*, 1993.

[16] Jian-Jia Chen, Mario Günzel, Peter Bella, Georg von der Brüggen, and Kuan-Hsun Chen. Dawn of the dead (line misses): Impact of job dismiss on the deadline miss rate. *arXiv preprint arXiv:2401.15503*, 2024.

[17] Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. Efficient computation of deadline-miss probability and potential pitfalls. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 896–901. IEEE, 2019.

[18] Tommaso Cucinotta and Dario Faggioli. An exception based approach to timing constraints violations in real-time and multimedia applications. In *International Symposium on Industrial Embedded System (SIES)*, pages 136–145, July 2010.

[19] Robert Ian Davis and Liliana Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for Real-Time systems. *LITES: Leibniz Transactions on Embedded Systems*, pages 1–53, 2019.

[20] José Luis Díaz, Daniel F García, Kanghee Kim, Chang-Gun Lee, L Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 289–300. IEEE, 2002.

[21] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 1(4):397–413, 1993.

[22] Dieter Fox. Adapting the sample size in particle filters through KLD-sampling. *The international Journal of robotics research*, 22(12):985–1003, 2003.

[23] Bernardo Villalba Frias, Luigi Palopoli, Luca Abeni, and Daniele Fontanelli. Probabilistic real-time guarantees: There is life beyond the iid assumption (outstanding paper). In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 175–186. IEEE, 2017.

[24] Anna Friebe, Filip Marković, Alessandro V Papadopoulos, and Thomas Nolte. Efficiently bounding deadline miss probabilities of Markov chain real-time tasks. *Real-Time Systems*, pages 1–48, 2024.

[25] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726–740, 2014.

[26] Fredrik Gustafsson. Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):53–82, 2010.

[27] Huang-Ming Huang, Christopher Gill, and Chenyang Lu. Implementation and evaluation of mixed-criticality scheduling approaches for sporadic tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(4s):1–25, 2014.

[28] Raj Jain and Imrich Chlamtac. The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Communications of the ACM*, 28(10):1076–1085, 1985.

[29] Steven G. Johnson. The NLopt nonlinear-optimization package. http://github.com/stevengj/nlopt.

[30] Dieter Kraft. Algorithm 733: TOMP–fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software*, 20:262–281, 1994.

[31] Angeliki Kritikakou, Claire Pagetti, Olivier Baldellon, Matthieu Roy, and Christine Rochange. Run-time control to increase task parallelism in mixed-critical systems. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 119–128. IEEE, 2014.

[32] Łukasz Kruk, John Lehoczky, Kavita Ramanan, Steven Shreve, et al. Heavy traffic analysis for EDF queues with reneging. *The Annals of Applied Probability*, 21(2):484–545, 2011.

[33] Karthik Lakshmanan, Shinpei Kato, and Ragunathan Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *2010 31st IEEE Real-Time Systems Symposium*, pages 259–268. IEEE, 2010.

[34] Irina Iulia Lupu and Joël Goossens. Scheduling of hard real-time multi-thread periodic tasks. In Sébastien Faucou, Alan Burns, and Laurent George, editors, *19th International Conference on Real-Time and Network Systems, RTNS '11, Nantes, France, September 29-30, 2011. Proceedings*, pages 35–44, 2011.

[35] Sorin Manolache, Petru Eles, and Zebo Peng. Schedulability analysis of applications with stochastic task execution times. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(4):706–735, 2004.

[36] Filip Marković, Pierre Roux, Sergey Bozhko, Alessandro V Papadopoulos, and Björn B Brandenburg. CTA: A correlation-tolerant analysis of the deadline-failure probability of dependent tasks. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 317–330. IEEE, 2023.

[37] Saranya Natarajan and David Broman. Timed C: An extension to the C programming language for real-time systems. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 227–239. IEEE, 2018.

[38] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.

[39] John K Ousterhout et al. Scheduling techniques for concurrent systems. In *ICDCS*, volume 82, pages 22–30, 1982.

[40] Andrea Parri, Mauro Marinoni, Juri Lelli, Giuseppe Lipari, et al. An implementation of a multiprocessor bandwidth reservation mechanism for groups of tasks. In *Proceedings of the 16th Real Time Linux Workshop, OSADL, Ed., Dusseldorf, Germany*, 2014.

[41] Paolo Pazzaglia, Claudio Mandrioli, Martina Maggio, and Anton Cervin. DMAC: Deadline-miss-aware control. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, page 1. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[42] Nicola Piccinelli. Libmpc++: A library to solve linear and non-linear MPC. https://github.com/nicolapiccinelli/libmpc.

[43] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems*, 49:404–435, 2013.

[44] Per Skarin. *Control over the cloud: Offloading, elastic computing, and predictive control*. PhD thesis, Lund University, 2021.

[45] Marco Spuri, Giorgio Buttazzo, and Fabrizio Sensini. Robust aperiodic scheduling under dynamic priority systems. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 210–219. IEEE, 1995.

[46] Václav Struhár, Silviu S. Craciunas, Mohammad Ashjaei, Moris Behnam, and Alessandro Vittorio Papadopoulos. Hierarchical resource orchestration framework for real-time containers. *ACM Transactions on Embedded Computing Systems*, 23(1), January 2024.

[47] Zelin Tong, Shareef Ahmed, and James H Anderson. Holistically budgeting processing graphs. In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 27–39. IEEE, 2023.

[48] Don Towsley and SS Panwar. *Optimality of the stochastic earliest deadline policy for the G/M/c queue serving customers with deadlines*. Citeseer, 1991.

[49] Amy R Ward. Asymptotic analysis of queueing systems with reneging: A survey of results for FIFO, single class models. *Surveys in Operations Research and Management Science*, 17(1):1–14, 2012.

[50] Ward Whitt. Fluid models for multiserver queues with abandonments. *Operations research*, 54(1):37–54, 2006.

[51] Ward Whitt. Stabilizing performance in a single-server queue with time-varying arrival rate. *Queueing Systems*, 81:341–378, 2015.

[52] Ward Whitt. Time-varying queues. *Queueing models and service management*, 1(2), 2018.

[53] Sisu Xi, Chong Li, Chenyang Lu, Christopher D. Gill, Meng Xu, Linh T.X. Phan, Insup Lee, and Oleg Sokolsky. RT-Open Stack: CPU resource management for real-time cloud computing. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 179–186, June 2015.

[54] Xue Yang and Lorenz T Biegler. Advanced-multi-step nonlinear model predictive control. *Journal of process control*, 23(8):1116–1128, 2013.

[55] Kevin Zagalo, Yasmina Abdeddaïm, Avner Bar-Hen, and Liliana Cucu-Grosjean. Response time stochastic analysis for fixed-priority stable real-time systems. *IEEE Transactions on Computers*, 72(1):3–14, 2022.

Chapter 12

Paper E
Resource Management for Stochastic Parallel Synchronous Tasks: Bandits to the Rescue.

Anna Friebe, Alberto Marchetti-Spaccamela, Tommaso Cucinotta,
Alessandro V. Papadopoulos, Thomas Nolte, and Sanjoy Baruah.
Under review.

Abstract

In scheduling real-time tasks, we face the challenge of meeting hard deadlines while optimizing for some other objective, such as minimizing energy consumption. Formulating the optimization as a Multi-Armed Bandit (MAB) problem allows us to use MAB strategies to balance the exploitation of good choices based on observed data with the exploration of potentially better options. In this paper, we integrate hard real-time constraints with MAB strategies for resource management of a Stochastic Parallel Synchronous Task. On a platform with M cores available for the task, $m \leq M$ cores are initially assigned. Prior work has shown how to compute a virtual deadline such that assigning all M cores to the task if it has not completed by this virtual deadline guarantees that the deadline will be met. An MAB strategy is used to select the value of m. A Dynamic Power Management (DPM) energy model considering CPU sockets and sleep states is described. Experimental evaluation shows that MAB strategies learn consistently suitable m, and perform well compared to binary exponential search and greedy methods.

12.1 Introduction

When scheduling real-time tasks, we often want to manage the use of resources to optimize some objective and also ensure that the tasks' deadlines are met. The objective may be, for example, minimizing energy consumption or maximizing the processor time available for other tasks in uninterrupted time periods. The problem can be modeled as a Multi-Armed Bandit (MAB) problem [30], where a decision maker repeatedly selects one of several fixed options, known as arms or actions; the impact of each action is not known a priori, and it is influenced by uncertainty. MAB strategies optimize for the average case while balancing exploration and exploitation to achieve the best expected result over a time period. The average behavior is often the main concern in resource management beyond strict timing requirements. For example, minimizing the energy consumption implies minimizing the integral of the power consumption over time, essentially minimizing the average power consumption. This paper gives an example of the integration of strict timing guarantees with MAB strategies to minimize energy consumption.

A scheduler initially assigns m cores to a DAG task but can use $M \geq m$ identical computing cores if needed. For a compute-bound task and a work-conserving schedule, [21] have shown that the task's deadline is met if it is assigned all M cores at a virtual deadline V. V depends on the initial number of cores m assigned and the worst-case properties of the DAG task. We rely on these results to ensure that deadlines are met and outline an MAB approach to select m over time, balancing exploration and exploitation. In [21], resource management strategies are evaluated, selecting m based on response time and m of the most recent task invocation. These strategies aim for the lowest possible m that keeps the response times below $V(m)$. The MAB approach differs in two important ways compared to these strategies. 1) Optimization is done with respect to the expected reward over time instead of the most recent observation. 2) The reward function is decoupled from the arm selection, allowing for optimizing the resource management towards any goal dependent on the arm response times.

The reward function is constructed to minimize the energy consumption under an energy model of a multicore system with CPU sockets and Dynamic

Power Management (DPM) with sleep states. The energy model is based on data from [27].

Contribution: The main contribution is the *MAB application* to improve average behavior while ensuring hard real-time guarantees. For this application, we define and use a *Stochastic Parallel Synchronous Task*, a special case of a DAG task but a generalization of the Parallel Synchronous Task [26]. We derive *response time bounds* for different initial core allocations. Each arm represents a choice of initial core allocation. The MAB is implemented as a bootstrap/ bag approximation [20] of Thompson sampling [30]. In the proposed partial-feedback MAB, information about unexplored arms is derived from arms that have been explored, using the response time bounds.

In the evaluation, the proposed MAB core allocation strategy is compared with an MAB not using the derived response time bounds, the Binary-Exponential Search (BES) strategy from [21] adapted to the energy model, and a greedy strategy based on the energy model. The evaluation is performed for several selected task structures with different computation time variances and deadlines.

Outline of the paper: A background on Multi-Armed Bandits is given in Section 12.2. In Section 12.3 related work regarding task models with precedence constraints, bandit scheduling and energy-aware scheduling is outlined. Notation, the task model, and scheduling along with definitions are presented in Section 12.4. In Section 12.5 the resource management problem is formulated (Section 12.5.1). Methods from [21] are introduced with an example (Section 12.5.2). The proposed partial feedback MAB and the response time bounds are presented in Section 12.5.3. The energy model for the reward function is outlined in Section 12.6, and the evaluation in Section 12.7. In Section 12.8 conclusions and future work are discussed.

12.2 MAB Background

A MAB problem is a reinforcement learning problem in which an algorithm makes decisions over time under uncertainty [30]. The algorithm selects one out of K possible actions, called **arms**, in each of T rounds. Each action generates a reward according to a fixed but unknown probability distribution, and

the goal is to maximize the total reward over the T rounds, the **horizon**. There are many applications of MAB approaches, including healthcare, finance, dynamic pricing, recommender systems, anomaly detection, and telecommunications [8].

A standard approach to comparing different MAB algorithms is the concept of regret. Here, the sum of rewards for an algorithm over a horizon of T rounds is compared with the sum of expected rewards when consistently choosing the arm with the highest possible expected reward. That is, with a fixed but unknown reward distribution $\mathcal{D}_k$ of each arm k, the mean reward of an arm is denoted $\mu_k = \mathbb{E}[\mathcal{D}_k]$. The highest possible mean reward is $\rho^{\uparrow} = \max_k \mu_k$. The regret $R(T)$ over horizon T of an algorithm that at each round i takes an action a_i leading to reward $\rho(a_i)$ is defined as:

$$R(T) = \rho^{\uparrow} \cdot T - \sum_{i=1}^{T} \rho(a_i) \tag{12.1}$$

Feedback from a chosen action can be structured into three types [30]. Bandit feedback provides the reward for the chosen arm and no additional information. In a complete feedback setting, the agent can retrospectively observe the reward for all arms. Partial feedback implies that further information is provided beyond the reward of the chosen arm. In our case, an arm's response time provides information about the task's properties that is useful for all arms.

The reward model can be i.i.d. (independent and identically distributed), where the rewards of each arm are drawn from the same probability distribution, independent of the round and previous actions and rewards. Other reward models include rewards chosen by an adversary or evolving according to a random process [30]. In our case, we consider i.i.d. rewards.

The arm choice at a round i is based on the current estimates of the mean rewards μ_k of each arm. Algorithms often consider confidence intervals of the mean rewards. One common algorithm, Successive Elimination, removes an arm a from consideration when the upper bound of the confidence interval for μ_a is lower than the lower bound of the confidence interval for the mean reward μ_b of some other arm b. Another common algorithm, UCB1, always selects the arm with the highest upper confidence bound on the mean reward. The intuition

is that upper confidence bound is high due to the arm being a good choice, or due to the arm being unexplored and having a large confidence interval.

Bayesian bandits use the concepts in Bayesian statistics and assume that an unknown quantity is sampled from a known distribution [30]. The expected reward is maximized over the distribution, referred to as a belief model. Before selecting an arm in round t, we refer to the prior distribution of the belief. After obtaining the reward, the belief model is updated, and we refer to the posterior distribution. The posterior can be used as a prior in the next round. An algorithm for arm selection in a Bayesian bandit is Thompson sampling. In each round, every arm is assigned a probability of selection equal to the probability that the arm is optimal, given the history of previous rounds. An equivalent formulation is the following: at each round, a reward is sampled from the prior distribution over the expected reward of each arm. The arm corresponding to the highest reward is selected.

In some cases, the posterior distribution can be derived as a closed-form expression from a conjugate prior of the same family. However, in many cases, the posterior belief model is approximated [6]. One such approximation method is Bagging or Online Bootstrap Thompson Sampling [6, 20]. In this approach, several bags or replicates estimate the mean reward of the arms from observations in the past. Each replicate is updated with a new observation with a certain probability, resulting in the bags having different histories and mean reward estimates. In this way, the set of bags estimates the belief distribution of the reward mean.

In Contextual Multi-Armed Bandit (CMAB) problems, some feature vector or context is observed prior to the decision, and reward distributions are different for different contexts. For example, in a recommender system, the context could be user demographic information.

In a restless bandit setting [31], the reward distributions associated with the arms, including non-selected arms, may change across rounds. The objective is to maximize the average reward over an infinite horizon.

This paper considers the traditional MAB setting. There is no context information prior to the arm choice, and each arm's reward distribution remains fixed, but our knowledge about them evolves.

12.3 Related Work

12.3.1 Task Models With Precedence Constraints

Modeling precedence constraints with DAGs is common, both for precedence constraints between different tasks [15] and for precedence constraints within the same task [5, 26]. In Graham's list scheduling [15], a ready task is selected for processing each time a processor is idle. A ready task is a task with no precedence constraints or fulfilled precedence constraints. Graham's list scheduling does not produce a sustainable schedule [9], but bounds for the response time differences are presented [15].

In the parallel synchronous task model [26], a task's jobs consist of sequential segments, each containing threads that can run in parallel.

Even richer DAG-based models have been developed and studied. In the conditional parallel DAG task model [19], parallel nodes are combined with nodes representing if-then-else clauses. The multi-DAG model [14] models different execution flows as separate DAGs.

Papadopoulos *et al.* [21] is the work most closely related to this paper, and we describe the necessary content in Sections 12.4 and 12.5.2.

12.3.2 Bandit Scheduling

Yu *et al.* [33] have proposed using Restless Bandits for stochastic deadline scheduling in a data center. Here, arms represent positions in the job queue, and selecting an arm is equivalent to processing the job at that position in the queue on one processor. The problem is shown to be indexable. Whittle's index policy is not optimal, but the gap-to-optimality is bounded. Chen *et al.* [11] used a measure of information freshness in a restless bandit setting to determine which states to update. In a network setting, Raghunathat *et al.* [23] selected packets for broadcasting with a similar restless bandit approach. Borkar *et al.* [7] have applied a restless bandit approach in which a queue is selected for packet transmission on a channel, considering costs modeling the delay constraints and transmission energy consumption. To the best of our knowledge, bandit scheduling has not been applied to DAG or synchronous task models. Most of the approaches in the literature use arms to represent packets or jobs. This

requires a restless bandit approach as the states of all packets/ jobs evolve, whether or not they are transmitted/ processed. In our work, an arm represents the number of cores initially assigned to a task. This enables a more simplistic MAB problem while the considered task model is more complex.

12.3.3 Energy-Aware Scheduling

In modern processors, the two main approaches to control energy consumption are Dynamic Voltage and Frequency Scaling (DVFS), in which the voltage and CPU frequency are lowered at times, and Dynamic Power Management (DPM), in which a number of different processor idle states (C-states[1]) are used [4]. Deeper idle states save more power by turning off additional components in the CPU, but they require higher wake-up latencies. As noted in [16], the potential advantages of DVFS over DPM are decreasing. One reason is that reduced transistor sizes lead to an increased proportion of leakage currents, as these are a quantum phenomenon [4].

Bambagini *et al.* [4] survey and discuss work on energy-aware scheduling for real-time systems. [29] presented a survey on energy-efficient multicore scheduling for hard real-time systems. Xie *et al.* [32] surveyed low-energy parallel scheduling algorithms focused on DVFS techniques. Additional works that focused recently on the use of DVFS and big.LITTLE architectures for real-time tasks can be found in [18]. The former work introduces BL-CBS, an extension of the Adaptive Partitioned EDF scheduler in [1], where each real-time task CBS server is dynamically placed on the most energy-convenient CPU, chosen among big or LITTLE ones, considering the impact of possible frequency changes, needed to preserve schedulability, on the power consumption of the whole affected island. The technique is further extended in [17] to schedule real-time DAG tasks.

The power P_{gate} of a single active gate is described as a function of the probability of gate switching α, the loading capacitance C_L, the supply voltage V, the clock frequency f, the short circuit current I_{sc} and the leakage current

[1] C0 is actually the name of the fully operational state of the CPU, when it is executing instructions, C1 is the first software-only idle state, whereas C2, C3, etc... are the deep idle states. For more details, see: `https://doc.opensuse.org/documentation/leap/archive/42.2/tuning/html/book.sle.tuning/cha.tuning.power.html`.

I_{leak} [4, 10, 29][2]:

$$P_{gate} = \alpha \cdot C_L \cdot V^2 \cdot f + V \cdot I_{sc} + V \cdot I_{leak} \tag{12.2}$$

An important part of the power savings of DVFS is the concurrent reduction in the supply voltage enabled by a reduced clock frequency [16]. In some work [22, 3, 4], the dynamic power component is assumed to be $P(f) = \beta \cdot f^{\delta}$, $2 \leq \delta \leq 3$. With today's low core voltages, the voltage-scaling window is reduced, resulting in a smaller benefit of the dynamic component [16]. A lower clock frequency also implies a longer computation time, limiting when a low-power idle state can be used.

Schone *et al.* [27] describe the per-core C-states of x86 processors, and Package C-states that can be entered if all the cores in a socket are in a sleep state. In these Package C-states, power demand is reduced further, for example, by partially disabling the last-level cache. Recent work by Antoniou *et al.* [2] have proposed an alternative architecture to significantly reduce the wake-up latency of sleep states while retaining most of the power savings.

Sleep state arbiters have been developed to minimize data center response times and latency while saving energy. Examples include feedback control [34] and machine learning [28]. In [12], sleep states are coordinated with request delays and voltage frequency scaling to reduce tail latency and energy consumption. Request tail latency is estimated using random variables.

Despite the reduced gains in DVFS techniques, these and hybrid methods combining DVFS and DPM are common in the literature. The energy model presented in Section 12.6 uses DVFS only as a complement to DPM for cores in sleep states. This model can be incorporated into the framework with virtual deadlines for DAGs developed in [21] with minimal modification.

12.4 System Model and Notation

The main notation used in the paper is listed in Table 12.1.

$\mathcal{U}(a, b)$ denotes a uniform distribution on the range $[a, b]$. We use the same notation for the continuous uniform distribution on $[0, 1]$ and discrete uniform distribution on an integer range.

[2] In [4] α also multiplies the second term.

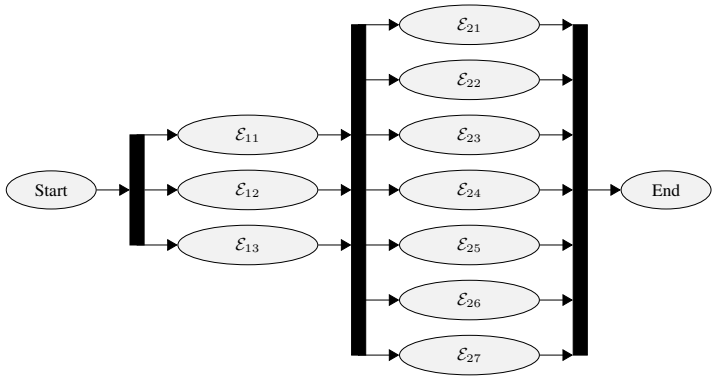


Fig. 12.1: A stochastic parallel synchronous task example.

12.4.1 Task Model

We consider a periodic task τ, generating a sequence of jobs J_i, $i \in \mathbb{N}$. Job arrivals are separated by the period p, and we refer to the arrival of J_i as round i. The task, referred to as a stochastic parallel synchronous task, has the internal structure of a parallel synchronous task [26]. That is, τ is composed of s sequential computation segments. There are u_j threads in the j-th segment of τ, and the total number of threads is U. The task structure is illustrated in Fig. 12.1.

Each segment ends in a synchronization point, so all threads in one segment must be completed before the next segment can begin its execution. In our task model, the execution time e_{ijk} of the k-th thread in job i's j-th segment is the outcome of an i.i.d. random variable $\mathcal{E}_{jk}$ with bounded support. The work w_i of job i is the sum of the execution times of all threads.

The main difference with respect to the model in [26] is the specification of thread execution times. In [26] the execution times of threads in a segment are specified by the worst case execution requirement of the task segment. In our model, execution times are outcomes of random variables. We assume that a fixed but unknown probability distribution specifies the execution requirement for a thread in a segment; different threads in the same segment may have different probability distributions.

The worst-case span L and work W of a DAG task are defined in [21], and we define them here for the stochastic parallel synchronous task. The worst-case span L of τ is a deterministic conservative estimate of the sum of the longest thread execution times from each segment as formalized in Eq. (12.3)

$$L \geq \sum_{j=1}^{s} \max_{k=1,\ldots,u_j} e_{ijk}, \forall i \tag{12.3}$$

W is the worst-case work of τ, a deterministic conservative estimate of the worst-case total computation time of all threads, as stated in Eq. (12.4).

$$W \geq \sum_{j=1}^{s} \sum_{k=1}^{u_j} e_{ijk}, \forall i \tag{12.4}$$

The assumption of bounded support for the execution time distribution of the k-th thread of the j-th segment $\mathcal{E}_{jk}$ of τ is necessary to assume that τ has deterministic worst-case work and span.

Let r_i denote the response time of J_i. The response time for any job should not exceed the relative deadline D. The deadline is implicit, $D = p$.

12.4.2 Scheduling

As in [21], jobs are scheduled on up to M identical cores. At the arrival, J_i is assigned m_i cores. Depending on m, a virtual deadline V is calculated according to Eq. (12.5).

$$V(m_i) = \left\lfloor \frac{M \cdot (D - L) - (W - L)}{M - m_i} \right\rfloor, m_i < M \tag{12.5}$$

If J_i is not completed at the time of $V(m_i)$, all M cores are assigned to J_i at this point. As shown in [21] this guarantees that J_i will meet its deadline.

When a job J_i is completed, the resulting response time r_i and total computation time w_i are reported, and this information is available to determine subsequent core assignments.

The guarantee that jobs meet their deadlines is based on the bound on the response time $r(m)$ of a DAG task job assigned m cores until completion in Eq. (12.6).

$$r(m) \leq L + \frac{W - L}{m} \tag{12.6}$$

This relies on the fact that the mean work of one of the cores and not on the critical path is at most $\frac{W-L}{m}$ [19], assuming no interference from other tasks, a constrained relative deadline ($D \leq p$) and a work-conserving schedule. $\frac{W-L}{m}$ is the latest possible point when one core is available to process the work on the critical path.

The threads within a task are scheduled according to list scheduling [15]; that is, the threads are ordered in a list. Threads ready to execute will be run in the order they are listed. We assume that the list starts with the threads of the first segment, followed by the threads of the second, third, and so on. This is without loss of generality - assume that in a list ordering LO, thread a of segment H precedes thread b of segment H', $H' < H$. Since thread a cannot start execution until all threads of H' have completed execution, it follows that moving thread a just after the last thread of segment H' in LO will give the same schedule.

We provide the definitions below for the *partial schedule*, *profile*, and the *ahead* relation of profiles of a stochastic parallel synchronous task with threads scheduled according to list scheduling.

Definition 12.4.1. *The partial schedule $S_i(t)$ of job J_i at time t, $0 \leq t \leq D$, specifies for each thread the total time the thread has been processed up to time t.*

The profile $Q_i(t)$ of a partial schedule $S_i(t)$ is the list $Q_i(t) = (q_1(t), q_2(t), \ldots, q_U(t))$ where $q_l(t)$, $l = 1, 2, \ldots, U$, represents the remaining processing time at time t after the job's arrival of thread l from J_i, with threads ordered according to the list scheduling priority.

Since the threads' processing times are unknown, the remaining thread processing times are unknown. Next, we introduce an ordering relation among profiles that corresponds to different partial schedules of the job at different points in time.

Definition 12.4.2. *The profile* $Q_i''(t'') = (q_1''(t''), q_2''(t''), \ldots, q_U''(t''))$ *is ahead of profile* $Q_i'(t') = (q_1'(t'), q_2'(t'), \ldots, q_U'(t'))$ *if* $q_a''(t'') \leq q_a'(t'), \forall a$. *We denote this as* $Q_i''(t'') \leq Q_i'(t')$.

We note that the ahead relation is transitive. If $Q_i''(t'') \leq Q_i'(t')$ and $Q_i'''(t''') \leq Q_i''(t'')$ then $Q_i'''(t''') \leq Q_i'(t')$.

We also note that if at time t' the partial schedule S_i' executes a thread from segment H', and at time t'' the partial schedule S_i'' executes a thread from segment H'' that follows H', then $Q_i''(t'')$ is ahead of $Q_i'(t')$.

12.5 Resource Management

In this section, the problem formulation is outlined in Section 12.5.1. A motivating example is presented in Section 12.5.2, along with description of methods from [21]. The proposed partial feedback MAB approach is outlined in Section 12.5.3.

12.5.1 Problem Formulation

We want to choose the number of cores m to initially assign to a stochastic parallel synchronous task τ as described in Section 12.4, that minimizes a regret related to the task's response time distribution for the arms. In other words, we want to choose the arm that maximizes the total reward over a specified time horizon, where the reward of a job depends on the response time, execution requirement, and core assignment. The exact structure of the task is unknown to the scheduler, only the worst-case work W and span L are known. The scheduler observes the response time and total work of jobs after their execution. The deadline is guaranteed to be met, given that M cores are assigned to the task at the virtual deadline.

12.5.2 Motivating Example and Methods from Related Work

In [21], the objective was to find the ideal initial assignment of cores, resulting in a response time as close as possible to the virtual deadline without exceeding it. We can reformulate this to the probabilistic case - assume the objective is

to find the initial assignment of cores, resulting in the average response time as close as possible to the virtual deadline without exceeding it. Let us outline and apply two of the methods evaluated in [21], namely the *binary search* and the *binary-exponential search*, to a task as defined in Section 12.4. We will discuss why these methods are unsuitable for the stochastic parallel synchronous task model.

The general structure of the algorithms is outlined in Algorithm 12.1. After some initialization, at each round a core allocation is selected. The next job is run with the selected allocation, and the resulting response time and work are observed. Some update is performed based on the observations and the selected allocation.

Algorithm 12.1: General structure for core allocation over a horizon.

Input: Horizon T
Output: Allocations $(m_1, \cdots, m_T)$, response times $(r_1, \cdots, r_T)$, work $(w_1, \cdots, w_T)$

```
Function CoreAllocatorHorizon(T):
    Init()
    for i ∈ 1 : T do
        m_i ← CoreAllocator()
        r_i, w_i ← RunTask(m_i)
        Update(m_i, r_i, w_i)
```

Example 12.5.1. *Our example task has $s = 2$ segments, the first segment has $u_1 = 8$ threads, and the second has $u_2 = 4$ threads. The number of cores m to assign is in the range $[1, M], M = 10$. The scheduler uses a relative deadline $D = 16$, worst-case work $W = 52$, and span $L = 8$ to calculate the virtual deadlines.*

Example 12.5.2. *In a deterministic version of Example 12.5.1, the threads in the first and second segments have the lengths $(2, 2, 2, 5, 5, 2, 2, 2)$ and $(1, 1, 3, 3)$ in scheduling order.*

Example 12.5.3. *In a stochastic version of Example 12.5.1, the threads in the first segment have a length of 2 with 0.75 probability and 5 with 0.25 probabil-*

Algorithm 12.2: Binary search initialization.

Input: Maximum number of cores M
Output: $(lo, hi]$ interval
1 **Function** `InitBS(`M`)`**:**
2 $lo \leftarrow 0$
3 $hi \leftarrow M$

ity. The threads in the second segment have a length of 1 *with* 0.5 *probability and* 3 *with* 0.5 *probability.*

Binary Search For Selecting m

We recall the binary search algorithm from [21], intended for tasks with unknown and constant typical workload. It maintains an interval $(lo, hi], lo < hi$. For a deterministic task, the interval contains the ideal m resulting in a response time as close as possible to the virtual deadline without exceeding it. Initially $lo = 0$ and $hi = M$ - Algorithm 12.2 is the Init-function in Algorithm 12.1. The core allocation-function is Algorithm 12.3, $m_1 = \lceil \frac{lo+hi}{2} \rceil$. In the following rounds, m_{i+1} is determined and the $(lo, hi]$ interval is updated from m_i and r_i, where i represents the round when J_i arrives. The algorithm for updating the interval is outlined in Algorithm 12.4. Note that although lo starts at 0, $m_i > 0, \forall i$ because of the ceiling operation and that hi is only assigned to previous values of m.

Algorithm 12.3: Binary search for selection of m at round i.

Input: $(lo, hi]$ interval
Output: Cores m_i
1 **Function** `CoreAllocatorBS(`r_i, m_i, lo, hi`)`**:**
2 $m_i \leftarrow \lceil \frac{lo+hi}{2} \rceil$

We apply the binary search allocation to the deterministic task in Example 12.5.2. The allocation m and the interval are shown in Fig. 12.3, and the scheduling for the different m are shown in the first row of Fig. 12.2. The bi-

Algorithm 12.4: Binary search update of $(lo, hi]$ interval.

Input: Response time r_i, cores m_i, $(lo, hi]$ interval
Output: Updated $(lo, hi]$ interval

```
1 Function UpdateBS(r_i, m_i, lo, hi):
2     if r_i > V(m_i) then
3         lo ← m_i
4     if r_i < V(m_i) then
5         hi ← m_i
```

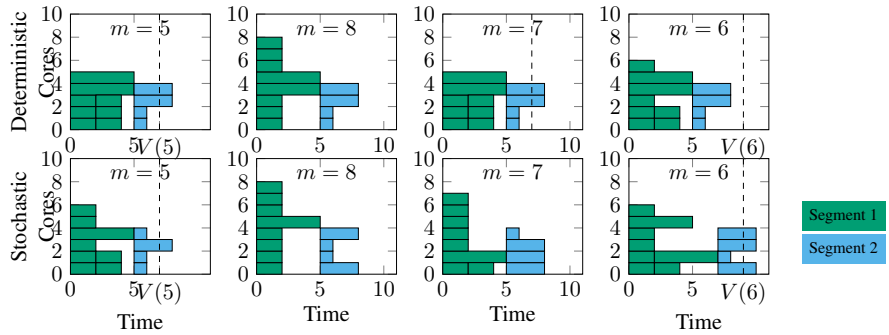


Fig. 12.2: Scheduling in the motivating example.

nary search starts with the interval $(lo = 0, hi = 10]$ and an initial $m_1 = 5$ corresponding to $V(5) = 7$ calculated from Eq. (12.5). In the first segment, three threads with length 2 and two with length 5 are scheduled, and the three remaining threads are scheduled at time 2, resulting in the completion time of 5 for the first segment. The response time $r_1 = 8$ is longer than the virtual deadline, and lo is updated to 5. The next $m_2 = \lceil (5 + 10)/2 \rceil = 8$, corresponding to $V(8) = 18$. The response time is 10, below $V(8)$, so hi is updated to 8, and the next $m_3 = 7$. Here we have $V(7) = 12$ and $r_3 = 8$, and a new $hi = 7$. This gives $m_4 = 6$, $r_4 = 8$, $V(6) = 9$, leading to $hi = 6$. From this stage, we will remain at $m = \lceil (5 + 6)/2 \rceil = 6$, the lowest m resulting in $r < V(m)$.

Next, we apply the binary search allocation to the stochastic task in Example 12.5.3. The evolution of the range and m selection is shown in Fig. 12.3

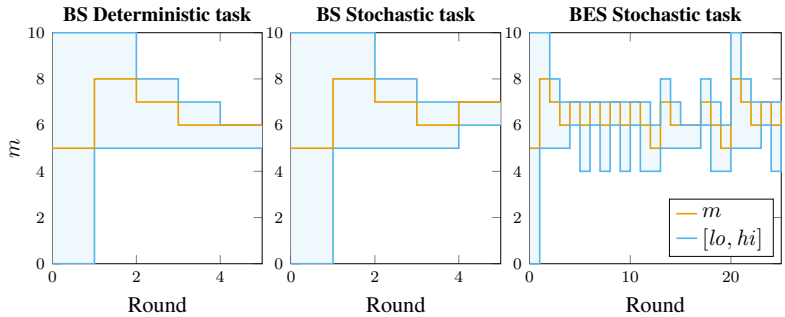


Fig. 12.3: The $[lo, hi]$ interval and the core allocation m in the motivating example task.

and the scheduling in the second row of Fig. 12.2. We first select $m_1 = 5$. At this time, the threads in the first segment have lengths $(2, 2, 2, 5, 2, 2, 2, 2)$, and in the second $(1, 1, 3, 1)$. The first segment has a completion time of 5, as all three threads in the first segment scheduled at 2 have length 2. The response time is therefore 8, longer than $V(5) = 7$, so lo is replaced with 5, and the new $m_2 = 8$. In this job instantiation, the threads in the first segment have lengths $(2, 2, 2, 2, 5, 2, 2, 2)$, , and the threads in the second segment have lengths $(3, 1, 1, 3)$, resulting in $r_2 = 8$. $V(8) = 18 > 8$, and $hi = 8$. Next, $m_3 = 7$ and the thread lengths of the first and second segment are $(2, 5, 2, 2, 2, 2, 2, 2)$ and $(3, 3, 3, 1)$ respectively. This results in a response time of $r_3 = 8 < V(7) = 12$, and $hi = 7$. In the next round, $m_4 = 6$, and the threads in the first segment have the lengths $(2, 2, 2, 2, 5, 2, 2, 5)$, and in the second segment thread lengths are $(3, 1, 3, 3)$. Now there is one thread of length 5 starting at 2, so the first segment completes at 7 and the response time is $10 > V(6) = 9$. leading to $lo = 6$. From now on, we will always select $m = 7$, although $m = 6$ is the lowest allocation where the average response time is lower than the virtual deadline.

Algorithm 12.5: Binary-exponential search initialization.

Input: Maximum number of cores M
Output: $(lo, hi]$ interval, inc, dec

```
1 Function InitBES(M):
2     lo ← 0
3     hi ← M
4     inc ← 2
5     dec ← 2
```

Binary-Exponential Search For Selecting m

The *binary-exponential search*(BES) from [21] enables adjustment of the interval $[lo, hi]$ from changes to the response times. For example, if the algorithm has converged, so that $m_i = hi$, but the response time is longer than the virtual deadline, $r_i > V(hi)$, it seems that we have converged to the wrong value. In this case hi will be increased by 2. If the response time in the next round is still longer than $V(hi)$, hi is increased by 4, with an exponential increase of hi or decrease of lo if response times are misaligned with the virtual deadlines of the $[lo, hi]$ interval. In addition to the $[lo, hi]$ interval, the BES maintains the variables inc and dec, initialized to 2 as outlined in Algorithm 12.5. These are used to enable an exponential increase of hi or decrease of lo if response times are misaligned with the virtual deadlines of the $[lo, hi]$ interval. The core allocation part is the same as for the *binary search* (Algorithm 12.3), and the modified update function is outlined in Algorithm 12.6.

Applying the BES algorithm to Example 12.5.3 leads to the interval being updated frequently, as illustrated in Fig. 12.3. The algorithm will select m near the ideal allocation, but the random variations in the response times will cause repeated updates of the search interval.

12.5.3 Partial Feedback Bayesian MAB

An MAB algorithm selects the initial number of cores m_i assigned to J_i. Each m that is valid represents an arm. The task must be schedulable for each valid arm, and a necessary schedulability condition is that the virtual deadline ac-

Algorithm 12.6: Binary-exponential search update of (lo, hi] interval.

Input: Response time r_i, cores m_i, (lo, hi] interval, interval update terms inc, dec
Output: Updated (lo, hi] interval and terms inc, dec

```
1  Function UpdateBES(r_i, m_i, lo, hi, inc, dec):
2      if r_i > V(m_i) then
3          if m_i = hi or r_i > V(hi) then
4              hi ← min(hi + inc, M)
5              inc ← inc · 2
6          lo ← m_i
7      if r_i < V(m_i) then
8          if r_i < V(m_i − 1) then
9              if m_i = lo − 1 or r_i < V(lo) then
10                 lo ← max(lo − dec, 0)
11                 dec ← dec · 2
12         hi ← m_i
13     if hi − lo ≤ 1 then
14         inc ← 2
15         dec ← 2
```

cording to Eq. (12.5) is non-negative. There may be further restrictions on valid m; the energy model presented in Section 12.6 and used in the evaluation is one such example.

The MAB models a current belief about each arm's reward and response time. The arms' response time distributions all depend on the properties of the parallel synchronous task. Generally, we expect a higher initial number of cores to result in a shorter response time. However, this is not always the case, as was already pointed out in [21]. In Line 9 we will discuss this further and provide response time bounds for different initial core allocations. The response time bounds are used in partial feedback to derive information about the reward of one arm from observations of another arm.

The MAB maintains κ bags (bootstrap replicates), each estimating the

mean reward for every arm. In Thompson sampling, an arm is selected with a probability equal to the probability that it is the best arm (that it has the highest mean reward) based on the history. We have chosen a bag implementation of the MAB, as it is applicable for general reward distributions and straightforward to implement and explain. There are many options for MAB implementations [30, 6], and many of these may perform better but may restrict the reward model or require additional steps, for example optimization. There are caveats [24, 25] with and benefits [13] of the bootstrap method. For example, a low number of bags tends to make the algorithm greedy [13].

The overall MAB algorithm has the general structure as Algorithm 12.1. In Algorithm 12.7, the initialization of the bags is performed. In each round, one of the bags is selected at random, and the arm m_i is chosen as the arm with the highest mean reward according to this bag, as outlined in $Algorithm$ 12.8. The estimate of mean rewards in a bag is described in detail in Line 9. We schedule J_i with m_i core assignment, then retrieve the response time r_i and total work w_i. In the update algorithm Algorithm 12.9 the reward ρ_i is calculated. For each bag B, we generate $nUpdatesBag$ as an outcome of *Poisson*(1), and update B with $nUpdatesBag$ copies of m_i, r_i, w_i, and ρ_i. This means that the expected number of observations in each bag after round i is i, but the bags will contain different parts of the history. A bag used to select m_i at round i is equally likely as any other bag to be updated with the resulting observation in one or more copies. Since the bags contain randomly selected parts of the history, the proportion of bags where an arm has the highest mean reward estimate is an approximation of the probability that the arm has the highest mean reward given the history. In this way, selecting a bag at random and taking the best arm given the bag's observations is an approximation of selecting an arm with the probability that it is the best given the history.

Response Time Bounds

In this section, we provide upper and lower response time bounds for a job scheduled with different number of cores initially. These are used in the partial feedback MAB, to obtain reward estimates for unexplored arms by using observations in explored arms, as outlined in Line 9. This is done with a counterfactual reasoning: for a job scheduled with m_a assignment and observed

Algorithm 12.7: MAB initialization.

Input: set of valid arms $validArms$, $horizon$, number of bags κ
Output: Initialized bags, valid arms.

```
1 Function InitMAB(validArms, κ):
2     bags ← κ empty bags
3     minArm ← min(validArms)
4     maxArm ← max(validArms)
```

Algorithm 12.8: MAB core allocation at round i.

Input: set of κ bags $bags$, min and max valid arm $minArm$, $maxArm$.
Output: Core allocation m_i

```
1 Function CoreAllocatorMAB(bags, minArm, maxArm):
2     B ← SampleFrom(bags))
3     if B.empty() then
4         m_i ← SampleFrom(U(minArm, maxArm))
5     else
6         m_i ← arg max_{m∈minArm:maxArm} EstMeanRew(B, m)
```

response time and work, what would the response time have been if it was instead scheduled with m_b assignment? In this way, we can avoid unnecessary exploration of non-optimal arms.

First, we derive a bound for the response time difference due to a lower initial number of cores having a smaller virtual deadline and M assignment at an earlier point. Next, we derive a bound on the response time difference resulting from different initial core allocations, based on [15]. We also provide response time bounds based on the job's work, since that does not change with the choice of allocation.

For example, consider a parallel synchronous task with two segments with four threads in each segment. For simplicity, we assume that the computation time required for each thread is deterministic. From Eq. (12.5) we conclude that $V(2) < V(3)$. As illustrated in Fig. 12.4, this can lead to a longer response time with a higher number of cores assigned initially. The response time difference is bounded by the difference of the virtual deadlines.

Algorithm 12.9: MAB update at round i.

Input: set of κ bags $bags$, arm m_i, observed response time, work r_i, w_i
Output: Updated bags.

```
1 Function UpdateMAB(bags, m_i, r_i, w_i):
2     ρ_i ← Reward(r_i, w_i, m_i)
3     for B ∈ bags do
4         nUpdatesBag ← SampleFrom(Poisson(1))
5         for k ∈ 1 : nUpdatesBag do
6             B.rewSum[m_i] ← B.rewSum[m_i] + ρ_i
7             Add(B.rt[m_i], r_i)
8             Add(B.work[m_i], w_i)
9             B.numObs[m_i] ← B.numObs[m_i] + 1
```

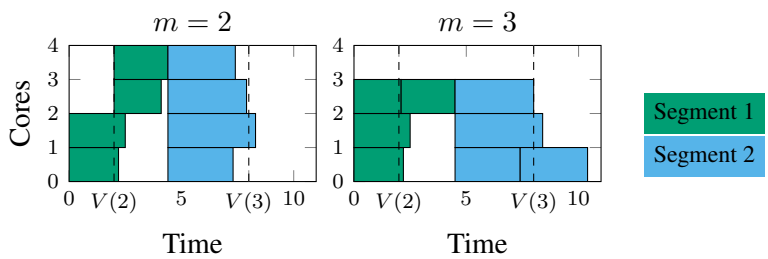


Fig. 12.4: A higher initial m resulting in a longer response time.

We will show in the following that Eqs. (12.7) to (12.9) hold, under the scheduling described in Section 12.4.2.

$$m_1 < m_2 \quad \Rightarrow \quad r_i(m_2) \leq r_i(m_1) + V(m_2) - V(m_1) \tag{12.7}$$

$$r_i(m_1) < V(m_1), m_1 < m_2 \quad \Rightarrow \quad r_i(m_2) \leq r_i(m_1) \tag{12.8}$$

$$r_i(m_2) > V(m_1), m_1 < m_2 \quad \Rightarrow \quad r_i(m_1) \geq V(m_1) \tag{12.9}$$

In the following, recall the definition of partial schedule S_i and profile Q_i of a job J_i in Definition 12.4.1, and the ahead relation of profiles from Definition 12.4.2.

Assumption 12.5.1. *The task model is as outlined in Section 12.4.*

Assumption 12.5.2. *Scheduling is with list scheduling, with the same list order for all arms and not modified during execution.*

Assumption 12.5.3. *A job J_i is scheduled with two different schedules, S_i' and S_i'', resulting in two different profiles Q_i' and Q_i''. At time t_0, profile $Q_i''(t_0)$ is ahead of $Q_i'(t_0)$, $Q_i''(t_0) \leq Q_i'(t_0)$. From time t_0 to time $t_1 > t_0$, S_i' assigns m_1 cores to J_i, and S_i'' assigns m_2 cores, $m_1 < m_2$.*

Assumption 12.5.4. *A job J_i is scheduled with two initial core assignments m_1 and m_2, $m_1 < m_2$. At the corresponding $V(m_1)$ and $V(m_2)$, respectively, the job is assigned M cores.*

Let us consider a single job J_i under different schedules.

Assume that job J_i starts execution at time 0 with m cores, resulting in a partial schedule $S_i(t)$ and profile $Q_i(t)$ at time t. Let $(e_1, e_2, \ldots, e_U)$ denote the execution times of each thread of J_i ordered in the list scheduling order of the threads.

Since thread processing times are unknown, the remaining processing times in the profile are also unknown. However the following observations on the profile $Q_i(t)$ hold:

- When a thread completes then its remaining processing time is 0: if thread a has completed execution then $q_a(t) = 0$;
- Since threads are processed using list scheduling the following holds: if $q_a(t) < e_a$ then all threads that precede a in list scheduling have started execution and, therefore, $q_b(t) < e_b$ for all $b, b < a$;
- If at time t partial schedule $S_i(t)$ executes a thread of segment H and thread a belongs to a segment that precedes H then a has completed execution, and, therefore, $q_a(t) = 0$;
- If at time t thread a has been processed for at least the same time interval in schedule $S_i(t)'$ compared to the schedule $S_i''(t)$ then $q_a'(t) \leq q_a''(t)$.

We now compare two different partial schedules $S_i'(t)$ and $S_i''(t)$ of J_i with different number of cores. Lemma 12.5.1 below will be applied in Proposition 12.5.2 to the part of the task execution taking place before $V(2)$ for both core allocations illustrated in Fig. 12.4.

Let $Q'(t) = (q'_1(t), q'_2(t), \ldots, q'_U(t))$ and $Q''(t) = (q''_1(t), q''_2(t), \ldots, q''_U(t))$ denote the profiles representing the remaining processing times when m_1 and m_2, $m_1 < m_2$, cores are used in in $S'(t)$ and $S''(t)$ respectively.

Lemma 12.5.1. *We use assumptions 12.5.1 to 12.5.3. At any time $t_0 \leq t \leq t_1$, $Q'_i(t) \geq Q''_i(t)$, i.e. $Q''_i(t)$ is ahead of $Q'_i(t)$.*

Proof. The proof is by induction. The statement is true at time $t = t_0$.

Assume that the statement is true at time $t-1$ and let A be the set of threads processed at t in schedule $S'_i(t)$ (that uses m_1 cores). If all threads in A are also processed at t in schedule $S''_i(t)$, then the claim is true at t by the inductive hypothesis.

Now assume that there exists a thread a, $a \in A$, that is not executed at t in S''_i; if thread a has already completed execution we have $q''_a(t) = 0$ and the claim holds at time t.

If thread a has not completed execution by time $t-1$ in S''_i, the segment executed at t in S''_i must be the segment to which a belongs because of the assumption that $Q'(t-1) \geq Q''(t-1)$. Then, at least m_2 threads should precede a in the list ordering that has not been completed. By the inductive hypothesis, these threads are also unfinished at time $t-1$ in the schedule that uses m_1 cores; since $m_2 > m_1$ this contradicts the assumption that a is executed at t when m_1 cores are available and the claim is true at time t. This completes the proof of the inductive step. □

Proposition 12.5.2. *We use assumptions 12.5.1, 12.5.2 and 12.5.4. At any time $0 \leq t \leq V(m_1)$, $Q'_i(t) \geq Q''_i(t)$, i.e. $Q''_i(t)$ is ahead of $Q'_i(t)$.*

Proof. Let $t_0 = 0$ and $t_1 = V(m_1)$. Then Proposition 12.5.2 follows from Lemma 12.5.1 because $Q'_i(0)$ and $Q''_i(0)$ are identical. □

Eqs. (12.8) and (12.9) follow directly from Proposition 12.5.2.

We now consider two partial schedules $S'_i(t')$ and $S''_i(t'')$ obtained using different number of cores and execution for different time intervals.

We are interested in comparing the completion times of $S'_i(t')$ and $S''_i(t'')$ when M of cores are used to complete $S'_i(t')$ ($S''_i(t'')$) after time t' (t''). Proposition 12.5.3 will be applied to the part of the task execution taking place after

$V(2)$ for $m = 2$ and after $V(3)$ for $m = 3$ in Fig. 12.4. $S'_i(t')$ ($S''_i(t'')$) uses M cores after time t' (t''). Let r'_i (r''_i) be the completion time of J_i under schedule $S'_i(t')$ ($S''_i(t'')$). The following Proposition shows that, if $Q''_i(t'')$, the profile of $S''_i(t'')$ at time t'', is ahead of $Q'_i(t')$ then $r''_i - t'' \leq r'_i - t'$, i.e. the time required to complete S''_i is not greater than the time necessary to complete S'_i when the same number of cores M is used.

Proposition 12.5.3. *We use assumptions 12.5.1, 12.5.2 and 12.5.4. If $t' \geq V(m_1)$, $t'' \geq V(m_2)$ and $Q''_i(t'')$, the profile of $S''_i(t'')$, is ahead of $Q'_i(t')$, the profile of $S'_i(t')$, then $r''_i - t'' \leq r'_i - t'$.*

Proof. Assume that $t' \leq t''$; (the proof for $t'' < t'$ is analogous).

It is sufficient to prove that, at any time instant t, $t \geq t'$, $Q'_i(t) \geq Q''_i(t + (t'' - t'))$. We prove it by induction. The statement is true at $t = t'$. We now assume it is true for $t - 1$, and we will prove that it holds at t.

Let A', A'' be the set of threads that are processed at t in schedule $S'_i(t)$ and at $t + (t'' - t')$ in $S''_i(t + (t'' - t'))$ respectively, and let I'' be the segment to which threads in A'' belong. If $A' = A''$, then the proposition is true at time t by the inductive hypothesis.

Now, assume a thread a, $a \in A' - A''$ exists. If a has already completed execution in $S''_i(t-1+(t''-t'))$ at $t-1+(t''-t')$ then its remaining processing time is 0 and the proposition holds at time t. We now observe that if $q''_a(t - 1 + (t'' - t')) > 0$, a must belong to segment I''. In fact, if a belongs to a segment that precedes I'' then l has completed execution in $S''_i(t-1+(t''-t'))$ and this implies that $q''_a(t-1+(t''-t')) = 0$. Since $Q'_i(t-1) \geq Q''_i(t-1+(t''-t'))$ a cannot belong to a segment that succeeds I''.

If a has not completed execution by time $t-1+(t''-t')$, it follows that there are M uncompleted threads in segment I'' at time $t + (t'' - t')$ that precede a in list ordering. By the inductive hypothesis, these threads have not completed execution in schedule $S'_i(t - 1)$, thus contradicting the assumption that a is scheduled at t in $S'_i(t)$.

This concludes the inductive step and the proof. □

Now, we are ready to prove Theorem 12.5.4, claiming that a potential increase in a job's response time when assigning a higher number of cores initially is bounded by the difference of the virtual deadlines.

Theorem 12.5.4. *We use assumptions 12.5.1, 12.5.2 and 12.5.4. Then* $r_i(m_2) \leq r_i(m_1) + V(m_2) - V(m_1)$.

Proof. Let $S'_i(V(m_1))$ ($S''_i(V(m_1))$) be the partial schedule at time $V(m_1)$ when m_1 (m_2) cores are used and let $Q'_i(V(m_1))$ ($Q''_i(V(m_1))$) be the profile at time $V(m_1)$. Since $m_2 > m_1$ Proposition 12.5.2 implies that $Q'_i(V(m_1)) \geq Q''(V_i(m_1))$.

We now observe that as $V(m_2) > V(m_1)$, $Q''_i(V(m_1)) \geq Q''_i(V(m_2))$.

By transitivity, it follows that

$$Q'_i(V(m_1)) \geq Q''_i(V(m_1)) \geq Q''_i(V(m_2))$$

By Proposition 12.5.3 $r_i(m_2) - V(m_2) \leq r_i(m_1) - V(m_1)$ follows. □

If we have a response time of a job with one m allocation, Eqs. (12.7) to (12.9) provide bounds for one end or the range of possible response times with another allocation. For the other end of the range, we will show that the following hold:

$$r_i(m_2) \leq r_i(m_1) \leq V(m_1) \Rightarrow r_i(m_2) \geq r_i(m_1) \cdot \frac{m_1}{m_2 + m_1 - 1} \tag{12.10}$$

$$r_i(m_1) > V(m_1) \Rightarrow r_i(m_2) \geq V(m_1) \cdot \frac{m_1}{m_2 + m_1 - 1} \tag{12.11}$$

$$r_i(m_1) > V(m_1) \cdot (m_1 + m_2 - 1)/m_1 \Rightarrow r_i(m_2) > V(m_1) \tag{12.12}$$

$$r_i(m_2) > V(m_1) \Rightarrow r_i(m_2) \geq r_i(m_1) - V(m_1) \cdot \frac{m_2 - 1}{m_1} \tag{12.13}$$

Eq. (12.10) is simply a reformulation of Theorem 1 from Graham [15], which proves a bound for the response time ratio for a DAG scheduled with list ordering and m_1 or m_2 processors. Eq. (12.11) follows because according to Theorem 1 from [15], the part of J_i completed at $V(m_1)$ with m_1 cores assigned can be completed earliest at $V(m_1) \cdot \frac{m_1}{m_2+m_1-1}$ with m_2 cores assigned.

We denote the response time of a job J with allocation m up until t from its arrival, thereafter M, with $r(J, m, t)$.

Theorem 12.5.5. *We use assumptions 12.5.1, 12.5.2 and 12.5.4. Then Eq.* (12.12) *holds.*

Proof. The response time if scheduling J_i with m_1 cores until completion, $r(J_i, m_1, \infty)$, is compared to $r_i(m_1)$. Because the profiles are identical at $V(m_1)$, we use $t_0 = V(m_1)$ in Lemma 12.5.1, that gives $r(J_i, m_1, \infty) \geq r_i(m_1)$.

We compare $r(J_i, m_1, \infty)$ to $r(J_i, m_2, \infty)$, and Theorem 1 from [15] gives that $r(J_i, m_2, \infty) \cdot \frac{m_1+m_2-1}{m_1} \geq r(J_i, m_1, \infty) \geq r_i(m_1)$.

Inserting $r_i(m_1) > V(m_1) \cdot \frac{m_1+m_2-1}{m_1}$ gives $r(J_i, m_2, \infty) > V(m_1)$. At $V(m_1)$ profiles are identical for scheduling with m_2 until completion or until $V(m_2)$, so $r_i(m_2) > V(m_1)$ □

Theorem 12.5.6. *We use assumptions 12.5.1, 12.5.2 and 12.5.4. Then Eq.* (12.13) *holds.*

Proof. The job J_i is split into two part-jobs denoted J_a and J_b. J_a is the threads and parts of threads completed at $V(m_1)$ when the job is scheduled upon m_2 cores. J_b is the remaining parts of threads and threads at this time.

Scheduling J_i with m_2 cores up until $V(m_2)$, and then with M cores is equivalent to scheduling J_a with m_2 cores, and immediately schedule J_b with m_2 cores up until $V(m_2) - V(m_1)$ and thereafter M. Therefore, we have:

$$r_i(m_2) = r(J_a, m_2, V(m_2)) + r(J_b, m_2, V(m_2) - V(m_1))$$

Clearly $r(J_a, m_2, V(m_2)) = V(m_1)$, so $r_i(m_2) = V(m_1) + r(J_b, m_2, V(m_2) - V(m_1))$.

From Proposition 12.5.2, $r(J_b, m_1, 0) \leq r(J_b, m_2, V(m_2) - V(m_1))$, giving:

$$r_i(m_2) \geq V(m_1) + r(J_b, m_1, 0)$$

Since the split is not done with m_1 assignment, we have:

$$r_i(m_1) \leq r(J_a, m_1, V(m_1)) + r(J_b, m_1, 0)$$

Combining these gives $r_i(m_2) \geq V(m_1) + r_i(m_1) - r(J_a, m_1, V(m_1))$.

We compare scheduling of J_a with m_1 cores until completion and scheduling it with m_1 cores until $V(m_1)$ and thereafter M cores. The profiles are identical at $V(m_1)$, and from Lemma 12.5.1 with $t_0 = V(m_1)$, $r(J_a, m_1, V(m_1)) \leq r(J_a, m_1, \infty)$.

Theorem 1 from [15] gives $r(J_a, m_1, \infty) \leq V(m_1)\frac{m_2+m_1-1}{m_1}$. Combining these results we have $r_i(m_2) \geq V(m_1) + r_i(m_1) - V(m_1)\frac{m_2+m_1-1}{m_1}$. □

Furthermore, we use the total computation time w_i of a given a job J_i to derive the following response time bounds that hold for all m. Eq. (12.14) states that the response time cannot be longer than the total computation time (the response time when J_i is scheduled on a single core). Eq. (12.15) states that the response time cannot be shorter than the time it takes to complete w_i if all assigned cores are busy from start to completion.

$$r_i \leq w_i, \forall m \tag{12.14}$$

$$r_i \geq \begin{cases} \frac{w_i}{m} & w_i \leq V(m) \cdot m \\ V(m) + \frac{w_i - V(m) \cdot m}{M} & w_i > V(m) \cdot m \end{cases} \tag{12.15}$$

Bag Mean Reward Estimates

The algorithm for estimating the mean reward of an arm m in a bag B is outlined in Algorithm 12.10. If there are observations from this arm in B, we simply take the mean observed reward in the arm as our estimate. However, if there are no observations, an observation from another arm is used to obtain the reward estimate. The closest lower or higher arm with observations in B is used as the source arm, with probability in relation to the number of observations each of these arms have. We know that in Algorithm 12.10 the B contains at least one observation for one arm, due to the check on Line 3 in Algorithm 12.8.

The reward estimate for a target arm from an observation in a source arm is obtained according to Algorithm 12.11 if the source arm m is lower than the target arm's. If the source arm has a higher m than the target arm, the reward estimate is obtained as described in Algorithm 12.12. In both these algorithms, an observation is randomly sampled from the source arm, and the response time and total work are retrieved. Now, we consider the possible response time range for this observation under the counterfactual scenario that the job was scheduled with the target arm, although it was in fact scheduled with the source

Algorithm 12.10: Estimate mean reward of an arm m in bag B.

Input: Bag B, arm m
Output: Estimated mean reward $\rho(m)$ for B

Function `EstMeanRew(`B, m`)`**:**
  $useThisArmProb \leftarrow 1$
  if $B.numObs[m] > 0$ **then**
    return $\frac{B.rewSum[m]}{B.numObs[m]}$
  $sumObs \leftarrow 0$
  if $\exists i < m$ *s.t.* $B.numObs[i] > 0$ **then**
    $loArm \leftarrow \max(i < m \text{ s.t. } B.numObs[i] > 0)$
    $sumObs \leftarrow sumObs + B.numObs[loArm]$
  if $\exists i > m$ *s.t.* $B.numObs[i] > 0$ **then**
    $hiArm \leftarrow \min(i > m \text{ s.t. } B.numObs[i] > 0)$
    $sumObs \leftarrow sumObs + B.numObs[hiArm]$
  $useLoArmProb \leftarrow 0$
  if $\exists i < m$ *s.t.* $B.numObs[i] > 0$ **then**
    $useLoArmProb \leftarrow \frac{B.numObs[loArm]}{sumObs}$
  $s \leftarrow$ `SampleFrom`$(\mathcal{U}(0,1))$
  if $s \leq useLoArmProb$ **then**
    return `SampleRewardFromLower`$(loArm, m)$
  else
    return `SampleRewardFromHigher`$(hiArm, m)$

arm. A response time range with the target arm is retrieved using the bounds derived in Line 9. A factor is uniformly sampled in [0,1], and used to determine where in the range the response time estimate for the target arm goes. A reward is calculated with m, the estimated response time and the sampled total work, and used as the mean reward estimate.

In Algorithm 12.11, the lower end of the range is obtained from Eqs. (12.10) to (12.12) and (12.15).The higher end of the range is obtained from Eqs. (12.7), (12.8) and (12.14). Comments are added in the pseudocode to relate lines to equations.

In Algorithm 12.12, the lower end of the target arm response time range is obtained from Eqs. (12.7) to (12.9) and (12.15). The higher end of the range is obtained from Eqs. (12.10), (12.13) and (12.14). Comments in the algorithm connect lines to equations.

12.5.4 Returning to the Motivating Example

Let us return to the motivating example in Section 12.5.2 and compare the BES algorithm with our proposed MAB approach. For this purpose, we construct a reward function ρ as in Eq. (12.16). For a selected arm m and the observed response time r, $\rho = 1$ if the response time is between $V(m-1)$ (or 0 for $m = 1$) and $V(m)$. Otherwise $\rho = 0$.

$$\rho(r, m) = \begin{cases} 1 & r \leq V(1), m = 1 \\ 1 & V(m-1) < r \leq V(m), m > 1 \\ 0 & otherwise \end{cases} \tag{12.16}$$

We use the proposed MAB approach with this reward function, $\kappa = 50$ bags, and the stochastic task described in Section 12.5.2. The resulting core allocations over 500 rounds compared to using the BES are shown inFig. 12.5. In the first round, $m = 3$ is randomly selected. In the next 10 rounds, arms in the range $5 - 8$ are chosen, with $m = 5$ selected 5 times, $m = 6$ selected 3 times, and $m = 7$ and $m = 8$ both selected once. After this point, $m = 6$ is chosen most of the time, and exploration is less frequent the more rounds we add.

Algorithm 12.11: Estimate reward in target arm from lower source arm sample.

Input: Bag B, source arm $srcArm$, target arm $tgtArm$
Output: Estimated reward $\rho(tgtArm)$ for B

Function `SampleRewardFromLower(`B, $srcArm$, $tgtArm$`)`**:**
  $sampleIdx \leftarrow$ `GetRandomIndex(`$B, srcArm$`)`
  $rt \leftarrow B.rt[srcArm][sampleIdx]$
  $w \leftarrow B.work[srcArm][sampleIdx]$
  `/* Eq. (12.10) */`
  $loRange \leftarrow rt \frac{srcArm}{srcArm+tgtArm-1}$
  `/* Eq. (12.12) */`
  if $loRange > V(srcArm)$ **then**
    $loRange \leftarrow rt - V(srcArm) \cdot \frac{tgtArm-1}{srcArm}$
  else
    `/* Eq. (12.11) */`
    if $rt > V(srcArm)$ **then**
      $loRange \leftarrow V(srcArm) \cdot \frac{srcArm}{srcArm+tgtArm-1}$
  `/* Eq. (12.15) */`
  $loRangeWork \leftarrow \frac{w}{tgtArm}$
  if $loRangeWork > V(tgtArm)$ **then**
    $loRangeWork \leftarrow V(tgtArm) + \frac{w-V(tgtArm)\cdot tgtArm}{M}$
  if $loRange < loRangeWork$ **then**
    $loRange \leftarrow loRangeWork$
  `/* Eq. (12.7) */`
  $hiRange \leftarrow rt + V(tgtArm) - V(srcArm)$
  `/* Eq. (12.8) */`
  if $rt < V(srcArm)$ **then**
    $hiRange \leftarrow rt$
  `/* Eq. (12.14) */`
  if $hiRange > w$ **then**
    $hiRange \leftarrow w$
  $u \leftarrow$ `SampleFrom(`$\mathcal{U}(0,1)$`)`
  $rtTgt \leftarrow loRange + u \cdot (hiRange - loRange)$
  return `Reward(`$rtTgt, w, tgtArm$`)`

Algorithm 12.12: Estimate reward in target arm from higher source arm sample.

Input: Bag B, source arm $srcArm$, target arm $tgtArm$
Output: Estimated reward $\rho(tgtArm)$ for B

Function `SampleRewardFromHigher(`B, $srcArm$, $tgtArm$`)`**:**
  $sampleIdx \leftarrow$ `GetRandomIndex(`$B, srcArm$`)`
  $rt \leftarrow B.rt[srcArm][sampleIdx]$
  $w \leftarrow B.work[srcArm][sampleIdx]$
  `/* Eq. (12.7) */`
  $loRange \leftarrow rt + V(tgtArm) - V(srcArm)$
  `/* Eq. (12.8) */`
  if $rt < V(tgtArm)$ **then**
    $loRange \leftarrow rt$
  else
    `/* Eq. (12.9) */`
    if $rt < V(srcArm)$ **then**
      $loRange \leftarrow V(tgtArm)$
  `/* Eq. (12.15) */`
  $loRangeWork \leftarrow \frac{w}{tgtArm}$
  if $loRangeWork > V(tgtArm)$ **then**
    $loRangeWork \leftarrow V(tgtArm) + \frac{w - V(tgtArm) \cdot tgtArm}{M}$
  `/* Eq. (12.10) */`
  $hiRange \leftarrow rt \cdot \frac{tgtArm + srcArm - 1}{tgtArm}$
  `/* Eq. (12.13) */`
  if $rt > V(tgtArm)$ **then**
    $hiRange \leftarrow V(tgtArm) \cdot \frac{srcArm - 1}{tgtArm} + rt$
  `/* Eq. (12.14) */`
  if $hiRange > w$ **then**
    $hiRange \leftarrow w$
  $u \leftarrow$ `SampleFrom(`$\mathcal{U}(0, 1)$`)`
  $rtTgt \leftarrow loRange + u \cdot (hiRange - loRange)$
  return `Reward(`$rtTgt, w, tgtArm$`)`

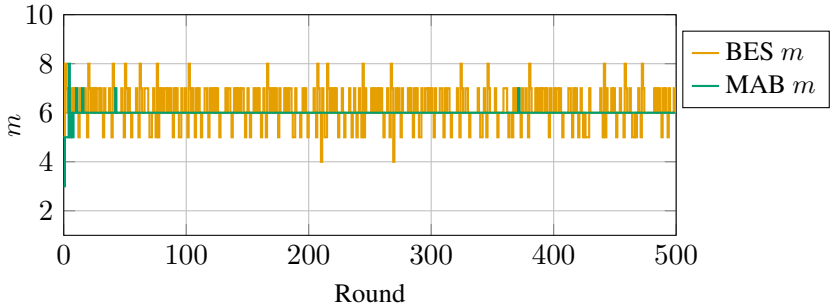


Fig. 12.5: Core allocation for the BES and MAB in the motivating example.

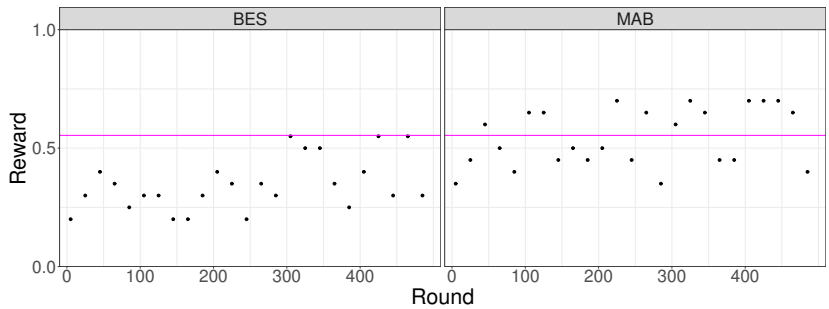


Fig. 12.6: Average rewards over each 20-round interval for the BES and MAB in the motivating example.

In Fig. 12.6, the rewards of the BES and MAB algorithms are shown. The reward in Eq. (12.16) is binary, so we show the average reward over each 20-round interval. The mean reward of $m = 6$ (the highest mean reward) is displayed in magenta. Another common way to evaluate MAB algorithms is the regret, Eq. (12.1). The difference between consistently choosing the arm with the highest expected reward and the algorithm choice is calculated over an interval or horizon (T in Eq. (12.1)). The regret for different horizons is shown in Fig. 12.7. Here, it is clear that the BES regret grows linearly with the horizon, while the MAB regret grows much slower once the algorithm has learned which arm is likely the best.

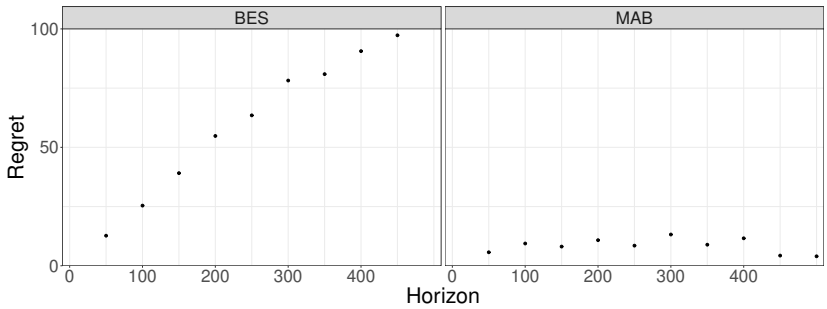


Fig. 12.7: Regrets over different horizons for the BES and MAB in the motivating example.

12.6 Energy Consumption for Reward Function

In this section, we show how to use the MAB approach to find initial core allocation m that minimizes the energy consumption over time. This is done by using an energy model that estimates energy consumption of jobs within the reward function. We emphasize that the MAB approach can also be used with other optimization goals.

The energy model is based on an existing microarchitecture with sleep states and the task model and scheduling from Section 12.4. In Section 12.6.1, we discuss how the sleep state latency affects the schedulability condition.

Consider a job arriving when m cores are in the running (C0) or halt (C1) state, and $M - m$ cores are in a deep sleep state. The wake-up latency of the sleep state is denoted as Δ_{RS}. If the parallel synchronous task has not completed at $V - \Delta_{RS}$, the $M - m$ sleeping cores need to be woken up. This model is a simplification: in a real system, the choice of m needs to be done Δ_{RS} prior to the job arrival, to ensure that cores are woken up if needed. This implies that the arm selection should be performed prior to this point, when the observations from the latest task invocation may be only partially complete (i.e., either the latest task completed and its execution time is known, or it is still running and its execution time is known with a small uncertainty bounded by Δ_{RS}). This is ignored in what follows, as the effect on the MAB performance is minor. Sleep states for a full socket can save power by disabling the last-level

cache. This could cause cold misses and longer response times. However, we would have the same concern when assigning all cores to the task at the virtual deadline. One of the assumptions stated in [21] is that the task is compute-bound.

The energy consumption is modeled according to the Sandy Bridge-EP (Xeon E5-2670) microarchitecture as described in [27], chosen because the power savings of the different states were documented here. Cores are distributed on n_s sockets with $n_{cs} = 8$ cores per socket. We let $M = n_s \cdot n_{cs}$, only this task is scheduled on n_s CPU sockets. Each core is in the running, halt, or sleeping state, corresponding to the CC0, CC1 and CC6 states. If all cores sharing a socket are in the sleep state, they enter the package sleep state, corresponding to PC6 in [27]. We assume that the scheduler can control when a core goes to and leaves the sleeping state. Linux allows for enabling or disabling individual sleep states either directly, or by specifying the per-core latency tolerance[3]. System-wide latency tolerance is set to allow for the use of deep sleep states. The scheduler restricts the use of deep sleep states for cores allocated to the task by temporarily setting a lower per-core latency tolerance. We do not require the cores to run at the highest possible frequency when executing the work of τ, but at a fixed frequency taken into consideration when determining W, L and schedulability.

The power consumption of a core in the running state is P_R. In the halt state, the power is P_H. In the sleep state, the power is P_S, and in the package sleep state the power is P_{SS}. Transitions between the halt and running state are instantaneous in the model, although a delay of less than $2\mu s$ was seen in [27]. Transitions to and from the sleep state are associated with a latency of Δ_{RS}, both for entering and exiting the sleep state. The average power requirement during this latency period is $P_{\Delta RS}$, and no work is processed during this time. The power consumption values for the different states are presented in Table 12.2, along with the transition latency. The power consumption values are inferred from the power savings compared to the running state presented in [27], except for the average power consumption during the transition to the sleep state, $P_{\Delta RS}$, as it is not presented in [27]. Therefore, we estimate it as follows. The residency times, that is the minimum time spent in the sleep

[3]https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/cpuidle

state leading to power savings, are documented in the Linux drivers[4]. Based on these and the latencies from [27], we estimate $P_{\Delta RS} = 7W$, equal to the power consumption in the running state.

Now we can describe how we model the power consumption over time from the arrival of a job J_i to its deadline, given the response time r_i and the total work w_i. Depending on the relation of D, r_i and V we outline 6 cases, illustrated in Fig. 12.8. The left column, Eq. (12.17), is the case where the job is completed sufficiently early, so it is advantageous to temporarily move the m cores into the sleep state and wake them up again before the next job arrival. The right column, Eq. (12.18), is the case where it is more advantageous to keep the m cores in the halt state.

The condition for the first column is $D - r_i > \Delta_{RS} \frac{P_{\Delta RS} - P_S}{P_H - P_S}$

For each of these, we have three cases, represented by the rows in Fig. 12.8. The first row, a), is the case when the job is completed in time so we don't need to wake up the $M - m$ cores, $r_i < V - \Delta_{RS}$. The second row, b), is when we wake them up, but it turns out they were not needed. $V - \Delta_{RS} \leq r_i \leq V$. The third row, c), is when he $M - m$ cores are used for completion of the job, $r_i > V$.

The full transition time Δ_{RS} is modeled at the time of going from the sleep state to the running state, although a smaller part of this is at the time of entering the sleep state.

The energy consumption $E_i(r_i, w_i, m)$ from the arrival until the deadline of J_i is calculated from the selected m, the response time r_i, and the total work w_i. The direct energy consumption from the computation is $w_i \cdot P_R$ appearing in all cases of Eqs. (12.17) and (12.18). The rest of the energy consumption comes from halt and sleep states and transitions between states. A core is in the halt state when it is assigned to the task by the scheduler, but all threads in the current segment are executing in other cores. The total time of cores in the halt state from an orange area in Fig. 12.8 is the size of the area minus w_i, found in all cases of Eqs. (12.17) and (12.18). A core is also in the halt state when there is no benefit in moving to the sleep state after the job has completed, shown as yellow areas in the right column of Fig. 12.8, and appearing in Eq. (12.18). The energy consumption while transitioning to and from the sleep

[4]https://github.com/torvalds/linux/blob/master/drivers/idle/intel_idle.c

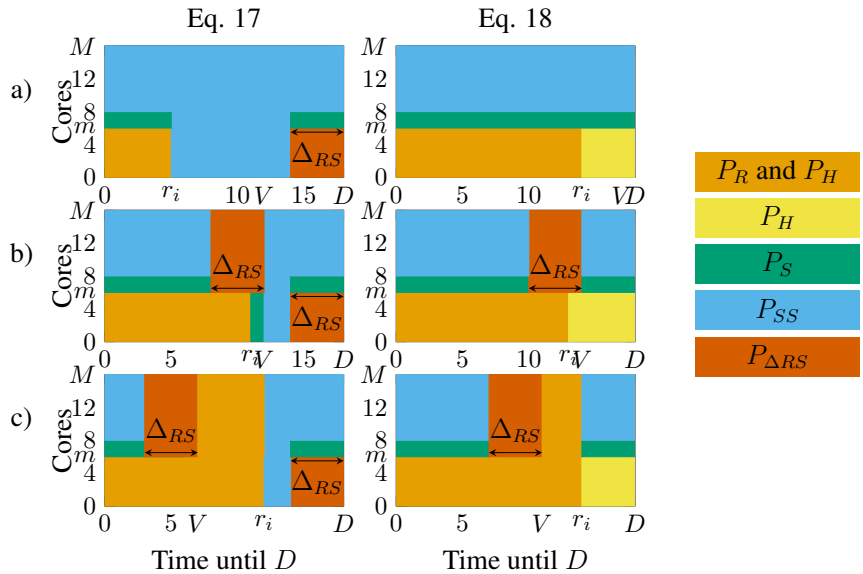


Fig. 12.8: Energy model illustration, $n_s = 2$, $n_{cs} = 8$, $M = 16$, $m = 6$.

state is determined by the red areas in Fig. 12.8, and found in Eqs. (12.17), (12.18b) and (12.18c). The energy consumption in the sleep state is shown as blue areas (full sleeping sockets) or green areas in Fig. 12.8, and found in all cases of Eqs. (12.17) and (12.18). Let $c_{s(M-m)}$ denote the number of cores in $M - m$ that make up full sockets, $c_{s(M-m)} = n_{cs} \cdot \lfloor \frac{M-m}{n_{cs}} \rfloor$. Analogously $c_{sm} = n_{cs} \cdot \lfloor \frac{m}{n_{cs}} \rfloor$ denotes the number of cores in m that make up full sockets. Let $c_{r(M-m)} = M - m - c_{s(M-m)}$ and $c_{rm} = m - c_{sm}$ denote the remaining cores in $M - m$ and m that share a socket with non-sleeping cores.

The energy consumption is calculated as:

$$\begin{aligned} E_i =& w_i \cdot P_R + (r_i \cdot m - w_i) \cdot P_H + \Delta_{RS} \cdot m \cdot P_{\Delta RS} + \\ & (r_i + \Delta_{RS}) \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \\ & (D - r_i - \Delta_{RS}) \cdot M \cdot P_{SS} \end{aligned} \tag{12.17a}$$

$$\begin{aligned} E_i =& w_i \cdot P_R + (r_i \cdot m - w_i) \cdot P_H + (V - r_i)(c_{sm} \cdot P_{SS} + c_{rm} \cdot P_S) + \\ & V \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \Delta_{RS} \cdot M \cdot P_{\Delta RS} + \\ & (D - V - \Delta_{RS}) \cdot M \cdot P_{SS} \end{aligned} \tag{12.17b}$$

$$\begin{aligned} E_i =& w_i \cdot P_R + (r_i \cdot m + (r_i - V) \cdot (M - m) - w_i) \cdot P_H + \\ & V \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \\ & \Delta_{RS} \cdot M \cdot P_{\Delta RS} + (D - r_i - \Delta_{RS}) \cdot M \cdot P_{SS} \end{aligned} \tag{12.17c}$$

$$\begin{aligned} E_i =& w_i \cdot P_R + (r_i \cdot m - w_i) \cdot P_H + (D - r_i) \cdot m \cdot P_H + \\ & D \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) \end{aligned} \tag{12.18a}$$

$$\begin{aligned} E_i =& w_i \cdot P_R + (r_i \cdot m - w_i) \cdot P_H + (D - r_i) \cdot m \cdot P_H + \\ & + (D - \Delta_{RS})(c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \Delta_{RS} \cdot (M - m) \cdot P_{\Delta RS} \end{aligned} \tag{12.18b}$$

$$\begin{aligned} E_i =& w_i \cdot P_R + (r_i \cdot m + (r_i - V) \cdot (M - m) - w_i) \cdot P_H + \\ & (V - \Delta_{RS} + D - r_i) \cdot (c_{s(M-m)} \cdot P_{SS} + c_{r(M-m)} \cdot P_S) + \\ & \Delta_{RS} \cdot (M - m) \cdot P_{\Delta RS} \end{aligned} \tag{12.18c}$$

We note that the highest possible energy consumption is $E^{\uparrow} = W \cdot P_R + M \cdot \Delta_{RS} \cdot P_{\Delta RS} + ((D - \Delta_{RS}) \cdot M - W) \cdot P_H$ and the lowest is $E^{\downarrow} = M \cdot D \cdot P_{SS}$. We use this to construct a reward function where rewards are in the interval $[0, 1]$, according to Eq. (12.19).

$$\rho(r_i, w_i, m) = \frac{E^{\uparrow} - E_i(r_i, w_i, m)}{E^{\uparrow} - E^{\downarrow}} \tag{12.19}$$

12.6.1 Schedulability Condition Considering Sleep State Latency

The $M - m$ cores that we need to use at the virtual deadline V may be in the sleep state, and we need to wake them up at $V - \Delta_{RS}$ to ensure they are

available on time. This means that with this energy model, a non-negative $V(m)$ according to Eq. (12.5) is not sufficient for schedulability, but valid m must fulfill the condition in Eq. (12.20).

$$V(m) = \frac{M \cdot (D - L) - (W - L)}{M - m} \geq \Delta_{RS}, m < M \tag{12.20}$$

12.7 Evaluation

The evaluation[5] is performed with task structures selected to highlight factors that affect the algorithm's performance. The reward function is based on the energy model described in Section 12.6, and specified in Eq. (12.19). The MAB algorithm outlined in Section 12.5.3 (**B_MAB**) is compared to an adapted **BES**, a **GREEDY** algorithm as outlined in Section 12.6.1, and a bandit feedback MAB algorithm that does not use the response time bounds to share information between arms (**NB_MAB**). Both MABs use $\kappa = 50$. In **NB_MAB**, arms without observations in a bag are selected with equal probability to the arm with the highest mean reward. That is, Algorithm 12.13 is used in place of Algorithm 12.8. The best arm with observations in the bag is added to a set with all arms without observations. An arm is selected at random from this set. Because Algorithm 12.10 is never called for an arm and bag without observations, the response time bounds are not used. We have not included a comparison with any DAG scheduling approach that requires knowledge about the DAG structure.

The task structures selected for the evaluation are listed in Table 12.3. Task structures 1, 3, and 5 have the same number of segments but varying degrees of parallelism within the segments. Task structures 2, 4, and 6 have the same degree of parallelism but with varying numbers of segments. Task structure 7 has four low parallelism segments and two high parallelism segments as the second and fifth segments. Task structure 8 is similar to task structure 7 but with both high parallelism segments at the end.

The execution time e_{ijk} of thread k in segment j of J_i is generated from a beta distribution, $Beta(\alpha = 2, \beta = 5)$, scaled and translated according to a

[5]Code and data will be made available upon acceptance.

Algorithm 12.13: NB_MAB core allocation at round i.

Input: set of κ bags $bags$, min and max valid arm $minArm$, $maxArm$.
Output: Core allocation m_i

Function `CoreAllocatorNB` ($bags$, $minArm$, $maxArm$) **:**
  $B \leftarrow$ `SampleFrom`($bags$)
  if *B.empty()* **then**
    $m_i \leftarrow$ `SampleFrom`($\mathcal{U}(minArm, maxArm)$)
  else
    $sampleArmsSet \leftarrow (m)$ s.t. $B.numObs[m] = 0$
    if $\exists m \notin sampleArmsSet$ **then**
      $m_{maxEst} \leftarrow \arg\max_{m \notin sampleArmsSet}$ `EstMeanRew`(B, m)
      $sampleArmsSet \leftarrow sampleArmsSet \cup (m_{maxEst})$
    $m_i \leftarrow$ `SampleFrom`($sampleArmsSet$)

setting β_γ and a translation β_Δ. e_{ijk} is sampled from $\mathcal{E}_{jk} \sim \beta_{\Delta jk} \cdot (1 + \beta_\gamma \cdot Beta(\alpha = 2, \beta = 5))$. The setting β_γ is varied in the experiments to explore the effect of thread execution times varying to different degrees between different jobs in the same task realization. $\beta_{\Delta jk}$ is drawn from a uniform distribution with the same width ($50\mu s$) for each thread, $\mathcal{U}(\beta_{\Delta\downarrow}, \beta_{\Delta\downarrow} + 50)$. The starting point of the uniform distribution $\beta_{\Delta\downarrow}$ is calculated according to Eq. (12.21) so that the expected value of thread execution times is the same ($150\mu s$) for all β_γ settings. This is to separate the effects of higher work and span from the effects of higher variance in the thread execution times within a realized sequence. The experiments generate tasks with $\beta_\gamma \in (0.1, 0.2, 0.4, 0.8, 1.6)$.

$$\beta_{\Delta\downarrow} = \left\lfloor \frac{150}{1 + \frac{2 \cdot \beta_\gamma}{7}} - \frac{50}{2} \right\rfloor \tag{12.21}$$

A task is generated for each task structure and β_γ setting. For each task, the worst case work W and span L are calculated as $W = \sum_{j=1}^{s} u_j \cdot (\beta_{\Delta\downarrow} + 50) \cdot (1 + \beta_\gamma)$ and $L = s \cdot (\beta_{\Delta\downarrow} + 50) \cdot (1 + \beta_\gamma)$. For each task structure, the maximum worst-case work and span over all β_γ settings are used to calculate virtual deadlines for all tasks.

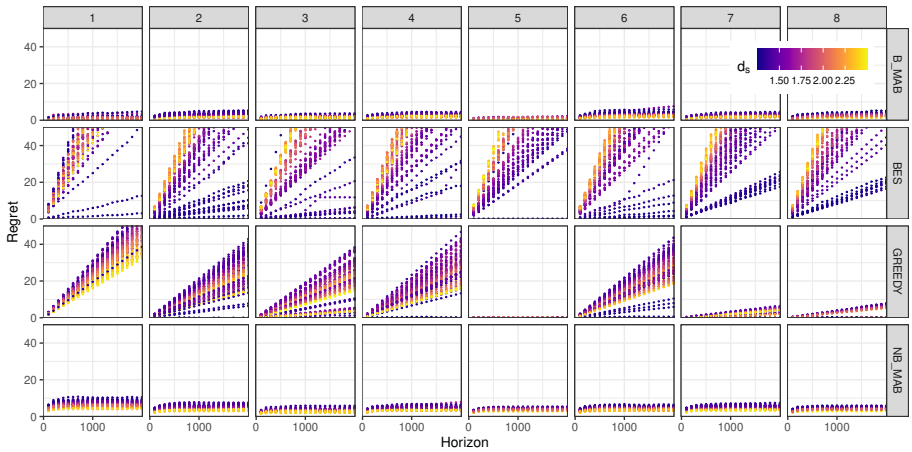


Fig. 12.9: Regrets for the methods over different horizon lengths for the task structures, with deadline configurations visualized.

The performance of the algorithms depends on the set deadline. A deadline set tightly compared to the schedulability condition leads to short virtual deadlines. This causes the BES to allocate more cores and affects the reward function for the MAB. Deadlines are generated as $D = \left(\frac{W+L}{M} + L + \Delta_{RS}\right) \cdot d_s$ from the schedulability condition given in Section 12.6.1. The factor d_s is randomly drawn from a uniform distribution $\mathcal{U}(1.25, 2.5)$ Every task is run with 20 deadline configurations, and the runs contain 2000 rounds.

The BES in the evaluation is adapted to use only valid m according to the schedulability condition Eq. (12.20), and aim for the lowest m that results in response times below $V(m) - \Delta_{RS}$ instead of below $V(m)$. The greedy method selects the lowest possible number of full sockets that fulfills the schedulability condition Eq. (12.20).

12.7.1 Results and Discussion

In Fig. 12.9, the regrets over different horizon lengths are displayed for the **B_MAB**, **BES**, **GREEDY**, and **NB_MAB** methods. Different deadline settings d_s are shown in different colors. InFig. 12.10, the same data is shown but with

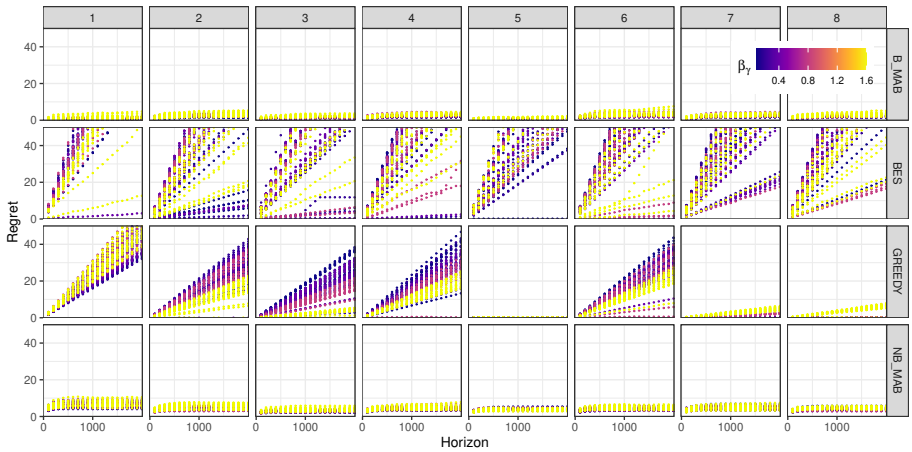


Fig. 12.10: Regrets for the methods over different horizon lengths for the task structures, with computation time variation configuration visualized.

coloring of the different configurations of β_γ that control the computation time variation between different threads.

It is clear that the regret of the **BES** and **GREEDY** methods grow linearly with the horizon, although the slope of the **BES** regret may change at points when the interval is updated. The regrets for the MAB algorithms grow much slower once the likely best arms are learned. There is no case where **B_MAB** has higher regret than 10 or **NB_MAB** has higher regret than 15. It is also clear that in some cases, **BES** and **GREEDY** have very low regret. These cases correlate with particular task structures and deadline factors, and for the **BES** case also low thread computation time variation. **BES** has lower regret for tighter deadlines, while the opposite is true for the MAB methods. **GREEDY** outperforms the other methods for task structure 5, with a large degree of parallelism. The MABs perform some initial exploration that comes with a cost to the regret, which is more pronounced for **NB_MAB**. Both MABs reliably find arms providing low regret in the long run for all tested task structures and deadline configurations.

The average regrets at step 2000 over all tasks and configurations are 18.9

for **GREEDY**, 98.9 for **BES**, 4.5 for **NB_MAB** and 2.1 for **B_MAB**. Using the response time bounds resulted in regrets less than half of those of **NB_MAB**. Due to the retrieved response time bounds, arms that cannot be optimal are not explored, reducing the regret.

To interpret the performance in terms of energy consumption, the total energy consumption at 2000 rounds with the energy model in Section 12.6 is calculated for each realization and each of the methods and for the arm that is best on average for the realization. The results are visualized in Fig. 12.11, where each black dot represents a realization in a bin, where bins have width $1J$. The red dots are the average energy consumption over the realizations for each method, that are also shown in Table 12.4, along with the ratios with the consumption using the best-average arm. **B_MAB** is within 0.5% of the optimal consumption, **NB_MAB** is within 1%, **GREEDY** within 4%, and **BES** within 25%. Statistical tests are performed with the sign test (binomial test). The number of realizations where the energy consumption is higher for one method than another is compared to the binomial distribution of 800 (the number of realizations) tests with success probability 0.5, the expected distribution if one method would be equally good as the other. The results show that **B_MAB** has lower energy consumption than **NB_MAB**, **NB_MAB** lower than **GREEDY**, and **GREEDY** lower than **BES**, all with p-value $< 10^{-15}$. The 95%-confidence interval of the binomial test success probability is $[0.994, 1]$ when comparing **B_MAB** to **NB_MAB**, showing that for a specific realization, **B_MAB** almost always outperforms **NB_MAB** slightly due to reduced initial exploration. The binomial test success probability is $[0.633, 1]$ when comparing **B_MAB** to **GREEDY** and $[0.612, 1]$ when comparing **NB_MAB** to **GREEDY**. There are realizations where **GREEDY** is optimal or close to optimal and outperform the MAB methods. Comparing to **BES**, the binomial test success probability is $[0.968, 1]$ for **B_MAB** and $[0.965, 1]$ for **NB_MAB**.

In Fig. 12.12, arm selections from one realization of each task structure are shown, along with the clairvoyant best choice for each job. The average best arm in the realization is shown as dashed magenta color. The **GREEDY** method finds the best allocation in the realizations shown for task structures 5 and 8. The **BES** method oscillates between different allocations. In the realizations for task structures 1, 3, and 6 these are near the optimal average choice.

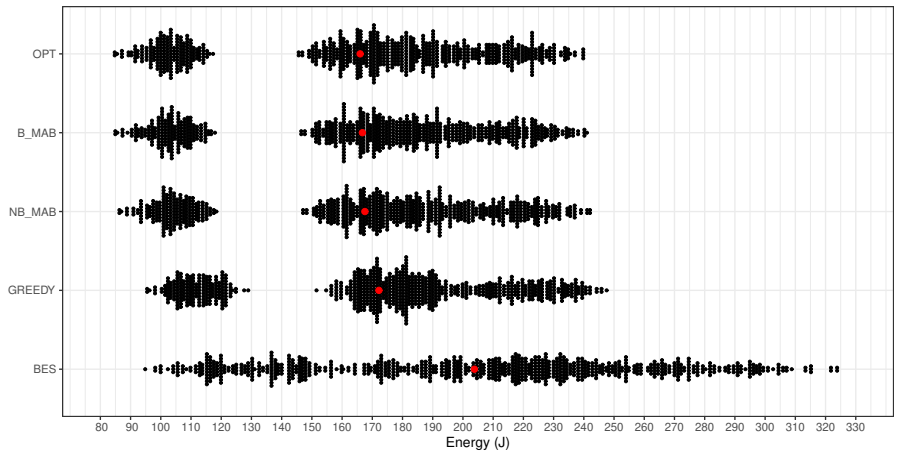


Fig. 12.11: Energy consumption at 2000 rounds over all realizations for the different methods compared to the best average arm (**OPT**) for each realization.

B_MAB has a lower amount of exploration than **NB_MAB**. For example the highest allocations in the realization of task structure 1 and the lowest allocations in the realizations of all task structures are almost never explored for **B_MAB**. Exploration for **B_MAB** occurs at later times, when **NB_MAB** no longer explores.

12.8 Conclusion and Future Work

This paper has integrated hard real-time constraints with an MAB resource management approach optimizing for the average case. Relying on previous work [21] to ensure that deadlines are met, an MAB approach is evaluated for assigning a suitable number of cores to a Stochastic Parallel Synchronous Task. A partial feedback MAB approach is proposed, utilizing response time bounds to obtain information for unexplored arms. The MAB approach has two main advantages compared to the methods evaluated in [21]. First, the MAB algorithm considers all observations, compared to the most recent observation only. Second, the reward function is decoupled from the arm selection, resulting in a

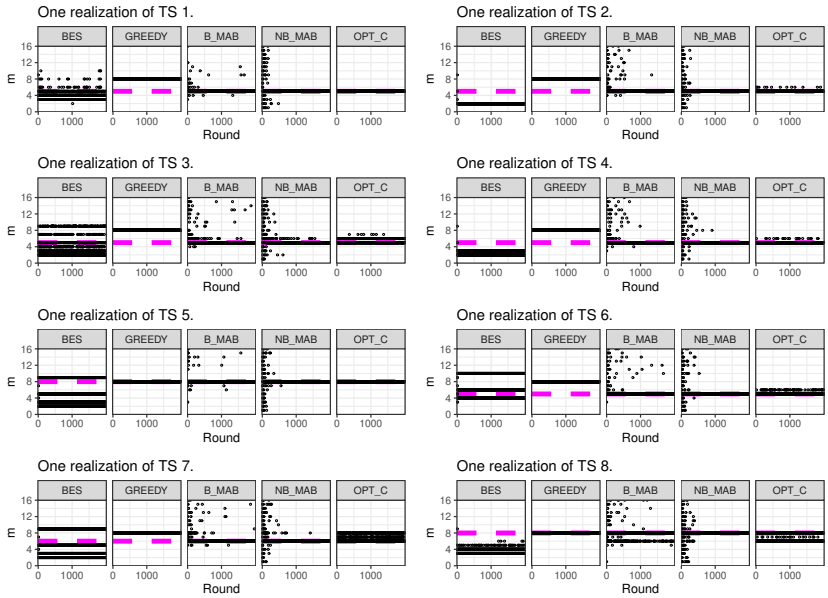


Fig. 12.12: Examples of arm selections for a few realizations along with the clairvoyant best arm for each job (**OPT_C**). The best average arm is dashed.

more versatile method. In the evaluation, the MAB approach is compared to the BES from [21], to a greedy method, and to a bandit feedback MAB not using response time bounds, for eight selected task structures over different settings for thread execution time variance and deadlines. Both MABs reliably find arm choices with low regrets in the long term for all task structures and settings, while the BES and greedy methods have low regret for certain combinations of task structure and deadline configuration. Using the response time bounds in a partial feedback MAB decreases the amount of initial exploration needed compared to the bandit feedback MAB.

The findings above show that an MAB approach is useful for resource management with optimization for the average behavior, can be integrated in a hard real-time context, and that the use of response time bounds for partial feedback improves the performance.

In future work, MAB integration in other real-time use cases will be explored. It would be interesting to investigate the case where the reward dependence on the response, work, and arm choice is unknown, for example, a reward taken from a power measurement. For systems with changing reward distributions, such as the tasks switching between different DAG structures in [21], CMAB or restless bandit approaches could be explored.

Table 12.1: Notation used in this paper.

Symbol	Description
τ, J_i	Task, job (at round) i of the task.
p	Task period.
s	Number of segments in a parallel synchronous τ.
u_j	Number of threads in the jth segment of τ.
U	Total Number of threads in τ.
$\mathcal{E}_{jk}$	Execution time random variable of the kth thread in the jth segment of τ.
e_{ijk}	Execution time of thread k in the jth segment of J_i.
L, W	The deterministic worst-case span, and work of τ.
r_i	The response time of J_i.
w_i	The total work of J_i.
D	The relative deadline of τ.
M	The maximum number of cores available for τ.
m_i	The number of cores assigned to J_i at its arrival.
$V(m)$	The virtual deadline of τ associated with m.
$S_i(t)$, $Q_i(t)$	The partial schedule and profile at time t after arrival.
n_s, n_{cs}	Number of sockets, and number of cores per socket.
P_X	The power consumption of a core in power state X.
Δ_{XY}	Transition latency in transitioning from power state X to Y and back to X.
$P_{\Delta XY}$	The average power consumption during Δ_{XY}.
c_{sx}, c_{rx}	The cores out of x cores that make up full sockets, and the remaining cores.
E_i	The modeled energy consumption from arrival to deadline of J_i.
$E^{\uparrow}$, $E^{\downarrow}$	The maximum and minimum possible energy consumption of a job.
T	Horizon (a specified number of rounds).
ρ	Reward.
B	Bag (bootstrap replicate) to estimate arm mean rewards.
κ	Number of bags.

P_R	P_H	P_S	P_{SS}	Δ_{RS}	$P_{\Delta RS}$
$7W$	$4W$	$2W$	$0.25W$	$40\mu s$	$7W$

Table 12.2: Core state power consumption and transition latency in the energy model.

TS	Description	s	u	TS	Description	s	u
1	Lo-u	5	(5, 5, 5, 5, 5)	2	Lo-s	4	(5, 10, 5, 10)
3	Mid-u	5	(10, 10, 10, 10)	4	Mid-s	6	(5, 10, 5, 10, 5, 10)
5	Hi-u	5	(15, 15, 15, 15, 15)	6	Hi-s	8	(5, 10, 5, 10, 5, 10, 5, 10)
7	Hi-u-in	6	(6, 16, 6, 6, 16, 6)	8	Hi-u-end	6	(6, 6, 6, 6, 16, 16)

Table 12.3: The evaluated task structures.

	OPT	**B_MAB**	**NB_MAB**	**GREEDY**	**BES**
Mean energy consumption [J]	166.0	166.7	167.5	172.2	203.8
Ratio with **OPT**	1	1.004	1.009	1.037	1.228

Table 12.4: Average energy consumption at 2000 rounds over all realizations for the different methods compared to the best average arm (**OPT**) for each realization.

Bibliography

[1] Luca Abeni and Tommaso Cucinotta. Adaptive partitioning of real-time tasks on multiple processors. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, page 572–579, New York, NY, USA, 2020. Association for Computing Machinery.

[2] Georgia Antoniou, Davide Bartolini, Haris Volos, Marios Kleanthous, Zhe Wang, Kleovoulos Kalaitzidis, Tom Rollet, Ziwei Li, Onur Mutlu, Yiannakis Sazeides, et al. Agile C-states: a core C-state architecture for latency critical applications optimizing both transition and cold-start latency. *ACM Transactions on Architecture and Code Optimization*, 2024.

[3] Hakan Aydin, Rami Melhem, Daniel Mosse, and Pedro Mejia-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings 13th Euromicro Conference on Real-Time Systems*, pages 225–232. IEEE, 2001.

[4] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):1–34, 2016.

[5] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. A generalized parallel task model for recurrent real-time processes. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 63–72. IEEE, 2012.

[6] Alberto Bietti, Alekh Agarwal, and John Langford. A contextual bandit bake-off. *Journal of Machine Learning Research*, 22(133):1–49, 2021.

[7] Vivek S Borkar, Gaurav S Kasbekar, Sarath Pattathil, and Priyesh Y. Shetty. Opportunistic scheduling as restless bandits. *IEEE Transactions on Control of Network Systems*, 5(4):1952–1961, 2017.

[8] Djallel Bouneffouf, Irina Rish, and Charu Aggarwal. Survey on applications of multi-armed and contextual bandits. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[9] Alan Burns, Sanjoy K. Baruah, et al. Sustainability in real-time scheduling. *J. Comput. Sci. Eng.*, 2(1):74–97, 2008.

[10] Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen. Low-power CMOS digital design. *IEICE Transactions on Electronics*, 75(4):371–382, 1992.

[11] Gongpu Chen, Soung Chang Liew, and Yulin Shao. Uncertainty-of-information scheduling: A restless multiarmed bandit framework. *IEEE Transactions on Information Theory*, 68(9):6151–6173, 2022.

[12] Chih-Hsun Chou, Laxmi N. Bhuyan, and Daniel Wong. μDPM: Dynamic power management for the microsecond era. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 120–132. IEEE, 2019.

[13] Dean Eckles and Maurits Kaptein. Bootstrap Thompson sampling and sequential decision problems in the behavioral sciences. *Sage Open*, 9(2):2158244019851675, 2019.

[14] José Carlos Fonseca, Vincent Nélis, Gurulingesh Raravi, and Luís Miguel Pinho. A multi-DAG model for real-time parallel applications with conditional execution. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1925–1932, 2015.

[15] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[16] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010*

international conference on Power aware computing and systems, pages 1–8, 2010.

[17] Agostino Mascitti and Tommaso Cucinotta. Dynamic partitioned scheduling of real-time DAG tasks on arm big.LITTLE architectures*. In *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, RTNS '21, page 1–11, New York, NY, USA, 2021. Association for Computing Machinery.

[18] Agostino Mascitti, Tommaso Cucinotta, Mauro Marinoni, and Luca Abeni. Dynamic partitioned scheduling of real-time tasks on arm big.LITTLE architectures. *Journal of Systems and Software*, 173:110886, 2021.

[19] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C. Buttazzo. Response-time analysis of conditional DAG tasks in multiprocessor systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 211–221, 2015.

[20] Nikunj C. Oza and Stuart J. Russell. Online bagging and boosting. In *International Workshop on Artificial Intelligence and Statistics*, pages 229–236. PMLR, 2001.

[21] Alessandro V. Papadopoulos, Kunal Agrawal, Enrico Bini, and Sanjoy Baruah. Feedback-based resource management for multi-threaded applications. *Real-Time Systems*, pages 1–34, 2022.

[22] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 89–102, 2001.

[23] Vivek Raghunathan, Vivek Borkar, Min Cao, and P. Roshan Kumar. Index policies for real-time multicast scheduling for wireless broadcast systems. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 1570–1578. IEEE, 2008.

[24] Donald B. Rubin. The Bayesian bootstrap. *The annals of statistics*, pages 130–134, 1981.

[25] Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on Thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

[26] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems*, 49:404–435, 2013.

[27] Robert Schöne, Daniel Molka, and Michael Werner. Wake-up latencies for processor idle states on current x86 processors. *Computer Science-Research and Development*, 30:219–227, 2015.

[28] Erfan Sharafzadeh, Seyed Alireza Sanaee Kohroudi, Esmail Asyabi, and Mohsen Sharifi. Yawn: A CPU idle-state governor for datacenter applications. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 91–98, 2019.

[29] Saad Zia Sheikh and Muhammad Adeel Pasha. Energy-efficient multicore scheduling for hard real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(6):1–26, 2018.

[30] Aleksandrs Slivkins. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[31] Peter Whittle. Restless bandits: Activity allocation in a changing world. *Journal of applied probability*, 25(A):287–298, 1988.

[32] Guoqi Xie, Xiongren Xiao, Hao Peng, Renfa Li, and Keqin Li. A survey of low-energy parallel scheduling algorithms. *IEEE Transactions on Sustainable Computing*, 7(1):27–46, 2021.

[33] Zhe Yu, Yunjian Xu, and Lang Tong. Deadline scheduling as restless bandits. *IEEE Transactions on Automatic Control*, 63(8):2343–2358, 2018.

[34] Xin Zhan, Reza Azimi, Svilen Kanev, David Brooks, and Sherief Reda. CARB: A C-state power management arbiter for latency-critical workloads. *IEEE Computer Architecture Letters*, 16(1):6–9, 2016.