Abstract

Real-time systems are central components in, e.g., industrial robots and automated guided vehicles, which integrate a wide range of algorithms with varying levels of timing requirements to achieve their functional behavior. Historically, in certain systems, these algorithms were deployed on dedicated single-core hardware platforms that exchanged information over a real-time network, while more recent designs have adapted an integrated architecture where these algorithms are executed on an embedded multi-core hardware platform. Meanwhile, the advantages provided by cloud and fog architectures for non-real-time applications have prompted discussions about the possibility of achieving similar benefits for systems such as industrial robot controllers. This thesis addresses a subset of challenges related to scheduling to facilitate this transition, and presents three main contributions aimed at improving online scheduling approaches in multi-server systems for applications with real-time requirements. First, an approach based on minimum parallelism reservations is proposed for scheduling sequential tasks in hierarchical multi-server systems with clairvoyant inputs, ensuring adherence to hard real-time requirements. Second, a framework is introduced that utilizes estimated processing times to enhance average throughput in distributed multi-queue multi-server systems while managing tasks with stochastic inputs and firm real-time requirements, thereby improving resource utilization. Finally, competitive algorithms are proposed that leverage estimated processing times to minimize average (modified) tardiness in centralized single-queue multi-server systems, addressing the scheduling of sequential tasks with arbitrary arrivals and soft real-time requirements. Collectively, these contributions establish a robust foundation for improving the

performance of real-time systems operating in increasingly complex environments characterized by dynamic workloads and varying resource availability.

Sammanfattning

Realtidssystem är centrala komponenter i till exempel industrirobotar och automatiserade självkörande fordon. I dessa system integrerar realtidssystem ett brett utbud av algoritmer med varierande nivåer av tidskrav för att uppnå sitt funktionella beteende. Historiskt sett har dessa algoritmer i vissa system distribuerats på dedikerade enkärniga hårdvaru plattformar, som utbytte information över ett realtids nätverk, medan nyare design har anpassat en integrerad arkitektur där dessa algoritmer körs på en inbäddad flerkärnig hårdvaruplattform. Fördelarna med nyare så kallade moln- och dimarkitekturer för andra typer av tillämpningar har föranlett diskussioner kring möjligheten att uppnå liknande fördelar för exempelvis styrsystem för industrirobotar, genom att gå från en inbäddad arkitektur till en molnbaserad arkitektur. Denna avhandling tar upp en del utmaningar relaterade till schemaläggning för att underlätta en sådan övergång, och presenterar tre huvudsakliga bidrag som syftar till att förbättra online schemaläggningsmetoder i multi-serversystem för applikationer med realtidskrav. För det första föreslås ett tillvägagångssätt baserat på minsta parallellitetsreservationer för att schemalägga sekventiella uppgifter i hierarkiska multiserversystem med klärvoajanta indata, vilket säkerställer att hårda realtidskrav följs. För det andra introduceras ett ramverk som använder beräknade behandlingstider för att förbättra den genomsnittliga genomströmningen i distribuerade multi-queue multi-server system samtidigt som uppgifter hanteras med stokastiska indata och fasta realtidskrav, och därigenom förbättra resursutnyttjandet. Slutligen föreslås konkurrenskraftiga algoritmer som utnyttjar uppskattade bearbetningstider för att minimera genomsnittlig (modifierad) försening i centraliserade enkelköiga multi-

serversystem, vilket tar itu med schemaläggning av sekventiella uppgifter med godtyckliga ankomster och mjuka realtidskrav. Tillsammans skapar dessa bidrag en robust grund för att förbättra prestandan och effektiviteten hos realtidssystem som arbetar i allt mer komplexa miljöer som kännetecknas av dynamiska arbetsbelastningar och varierande resurstillgänglighet.

Acknowledgment

I would like to thank my PhD advisors Thomas Nolte, Alessandro Papadopoulos and Saad Mubeen for their constant encouragement and positive feedback that made this endeavor rather fulfilling. I would also like to thank Anders Wall, Fredrik Hedenfalk and Daniel Eriksson, for their continuous support during my time at ABB. I am glad to have had insightful discussions with Per Willför and Mikael Norrlöf, which deepened my understanding of the challenges in software engineering for industrial robots. I would also like to appreciate the support I received from my amazing colleagues, Daniel Sehlberg, Nicklas Larsson, Kjetil Erga, Daniel Watson and Erik Dahlberg!

I cherish the memorable moments spent at the university, particularly with Zeinab Bakhshi Valojerdi, Taufik Akbar Sitompul, Lan Van Dao, Nitin Desai and Václav Struhár. Together, we engaged in enriching discussions, delving into research ideas as well as candid conversations about our personal lives, and exploring diverse perspectives on histories, cultures, and traditions.

I am certainly indebted to my family for their support and understanding over the years. To Naveera, for her unwavering patience and support throughout this journey.

This work is supported by ABB robotics and has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and the industrial postgraduate school Automation Region Research Academy (ARRAY++) funded by The Knowledge Foundation.

List of Publications

Papers included in this thesis[1]

Paper A: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *Multi-processor scheduling of elastic applications in compositional real-time systems.* In the Journal of Systems Architecture, Volume 122, 2022.

Paper B: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *Evaluating dispatching and scheduling strategies for firm real-time jobs in edge computing.* In the proceedings of the IEEE 49th Annual Conference of the IEEE Industrial Electronics Societ (IECON), 2023.

Paper C: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *Dispatching deadline constrained jobs in edge computing systems.* In the proceedings of the IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), 2023.

Paper D: Shaik Mohammed Salman, Van-Lan Dao, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *Scheduling firm real-time applications on the edge with single-bit execution time prediction.* In the proceedings of the IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC), 2023.

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

Paper E: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *Taming tardiness on parallel machines: online scheduling with limited job information.* MRTC Report, MDU 2024.

Related publications, not included in this thesis

Shaik Mohammed Salman, Vaclav Struhar, Alessandro V. Papadopoulos, Moris Behnam, and Thomas Nolte. *Fogification of industrial robotic systems: research challenges.* In Proceedings of the Workshop on Fog Computing and the IoT, 2019.

Shaik Mohammed Salman, Taufik Sitompul, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *Fog computing for augmented reality: trends, challenges and opportunities.* In the proceedings of the IEEE International Conference on Fog Computing (ICFC), 2020.

Shaik Mohammed Salman, Struhár, V., Bakhshi, Z., Dao, V. L., Desai, N., Papadopoulos, A. V., Nolte, T., Karagiannis, V., Schulte, S., Venito, A., & Fohler, G. *Enabling fog-based industrial robotics systems.* In the proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2020.

Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *A systematic migration methodology for complex real-time software systems.* In the proceedings of the IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC), 2020.

Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *A systematic methodology to migrate complex real-time software systems to multi-core platforms.* In the Journal of Systems Architecture, Volume 117, 2021.

Shaik Mohammed Salman, Saad Mubeen, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte. *Scheduling elastic applications in composi-*

tional real-time systems. In the proceedings of 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2021.

Van-Lan Dao, and Shaik Mohammed Salman. *Deep neural network for indoor positioning based on channel impulse response.* In the proceedings of 27th International Conference on Emerging Technologies and Factory Automation (ETFA), 2022.

Contents

I

Thesis

Chapter 1

Introduction

Systems such as industrial robots and automated guided vehicles rely on algorithms whose functional correctness depends on the availability of computed results within certain time intervals [1]. Such systems are said to have real-time requirements and are referred to as real-time systems. Software that manages systems with real-time requirements has been historically deployed on a network of dedicated single-core hardware devices. More recent designs embrace an integrated multiprocessor deployment architecture. In both approaches, the hardware devices are part of the physical system (embedded) or are in proximity to the system they control and the environment in which the system operates.

On the other hand, the cloud computing [2] paradigm provides computing capabilities through large data centers which often are geographically distant from end users. Despite this distance, it has transformed the ability to scale and deploy software for a broad set of businesses through platform-as-a-service and infrastructure-as-a-service models. This has led to considerable interest among academics and the industry in investigating and replicating the benefits offered by cloud computing for use cases that are dependent on embedded architectures.

For instance, architectural paradigms such as fog computing have been developed that intend to provide resources in proximity to end users to provide low response times while acting as gateways to centralized cloud resources [3]. Realizing the benefits of such architectures requires addressing a myriad of

technical challenges [4, 5, 6]. These include developing protocols that ensure predictable communication latencies [7], virtualization techniques that ensure isolated access to hardware resources [8], orchestration approaches that improve utilization [9] and scheduling techniques that can satisfy computational latency requirements of real-time applications [10].

In this context, this thesis focuses on a subset of challenges related to scheduling techniques to satisfy certain real-time requirements, taking into account on-demand availability and elasticity of computational resources. Specifically, the thesis provides:

> ***Online*** *algorithms that satisfy* ***hard*** *real-time requirements of* ***elastic*** *applications in clairvoyant settings, improve* ***throughput*** *in non-clairvoyant settings for* ***firm*** *real-time applications and reduce average modified* ***tardiness*** *in non-clairvoyant settings for* ***soft*** *real-time applications.*

The design and evaluation of these algorithms is explored in detail in the papers included in this thesis. To guide the reader, the following section outlines the thesis structure.

Thesis outline: This thesis consists of two parts. Part I provides an overview of the thesis and is organized as follows. Chapter 2 provides background information on scheduling, while Chapter 3 presents an overview of the current state-of-art closely related to the work in this thesis. Chapter 4 discusses the research challenges addressed in this thesis. Chapter 5 summarizes the research methods used to develop the contributions presented in the thesis. Chapter 6 provides an outline of the included papers and their contributions. Lastly, Chapter 7 summarizes the thesis and introduces potential directions for future work. Part II of the thesis consists of the included papers.

Chapter 2

Background

As the work in this thesis addresses scheduling challenges, this chapter provides the relevant background with an overview of scheduling concepts (Section. 2.1) together with the considered multi-server configurations (Section. 2.2) and the variants of the online scheduling problem that depend on the knowledge of job processing times (Section. 2.3).

2.1 Scheduling Theory

Scheduling theory broadly deals with the problem of providing sufficient resources (*servers*) to requesters (*jobs*) such that certain performance criteria are satisfied. Addressing this problem involves controlling the order and the duration for which the jobs are allowed to access servers. Variants of this problem are investigated via tools such as mathematical modeling, analysis, and simulations. The parameters contributing to the different variants of this problem include:

- Number of servers and their types.
- Job characteristics such as their processing times and release times.
- The ability to preempt and migrate the jobs among the different servers.
- Knowledge of job characteristics at the time of decision making.

- Performance objectives such as minimization of completion times, flow times, tardiness, and throughput maximization.

Depending on the parameters and the objective, a decision-making entity, commonly referred to as a *scheduler*, assigns, at any time t, jobs to servers according to a scheduling policy that determines the job order and the server access duration. Scheduling policies may allow preemption and possibly migration, i.e., jobs can be paused and resumed later and possibly on a different server. Depending on the availability of the complete knowledge of the input job instance before processing begins, scheduling policies can be offline or online. A scheduling policy is optimal for a given problem instance if it provides the best possible solution while satisfying all the associated constraints. An approximate scheduling policy offers a feasible solution that may not be optimal but is within a bounded factor of an optimal solution.

Offline Scheduling: In the offline version of a scheduling problem, information about an input job instance is assumed to be available before executing the first job. This information includes knowledge of processing times and the arrival times of all jobs. The goal is to utilize this information to create an optimal or approximate schedule that establishes a mapping between jobs, servers, and the time intervals during which a server is made available to a specific job [11]. This problem is commonly addressed in application areas such as distributed embedded systems [1]. Solution techniques include constraint programming [12] and meta-heuristics like simulated annealing [13].

Online Scheduling: In the online version of a scheduling problem, jobs arrive over time. Schedulers assign jobs to servers and determine the processing order based on the information available at the time of decision making. They make no assumptions about future jobs and have no information about the future jobs [14]. Typical decision-making time points include job arrival times and job completion times with the possibility of additional decision time points depending on the algorithm and problem instance [15, 16]. Online scheduling finds applications in computer systems such as cloud computing and real-time embedded systems.

Competitive Ratio: The performance of an approximate solution is commonly measured as the ratio between the performance of an approximate solution and that of an optimal solution. For online problems, an often-used measure is the online competitive ratio.

It is defined as the worst-case ratio between the performance of the online algorithm and the optimal offline algorithm [17]. For a minimization problem, the online competitive ratio c is defined as:

$$c = \sup_{\sigma} \frac{C_{\text{online}}(\sigma)}{C_{\text{optimal}}(\sigma)}$$

where $C_{\text{online}}(\sigma)$ is the cost incurred by the online algorithm for input sequence σ, and $C_{\text{optimal}}(\sigma)$ is the cost incurred by the optimal offline algorithm for the same input sequence. An algorithm with an online competitive ratio of c is said to be c-competitive.

2.2 Multi-Server Configurations

The work in this thesis, in most cases, focuses on online scheduling in a parallel server environment in which all the servers are homogeneous; that is, a job takes the same amount of processing time to finish on any server. Since several multi-server configurations are considered in this thesis, an overview is provided below.

Centralized Single-Queue Multi-Server Systems: This configuration is a fundamental model in scheduling [18] involving the availability of multiple parallel servers. Here, incoming jobs either preempt an existing job or wait in a centralized single queue. At any given time, each server may process at most one job. A preempted job is returned to the queue and may resume its execution on possibly a different server. Some servers may remain idle if there are fewer jobs in the queue than there are servers.

Distributed Multi-Queue Multi-Server Systems: This configuration models computer systems in which each server has its own queue [18]. Here, incoming jobs are assigned to a server as soon as they arrive. A scheduling policy

at each server determines the order in which the assigned jobs waiting in their respective queues are processed. Additionally, preempted jobs are retained in the same queue as the server on which they were preempted and resume their execution later on the same server.

Hierarchical Multi-Server Systems: This configuration models systems that consist of at least one dedicated server and, at most, one fractional server. In a hierarchical scheduling framework, the computational needs of an application are abstracted by a single interface that specifies the computational time to be reserved along with the time period in which it should be provisioned [19]. The reserved computing time can be made available to the application through various reservation approaches such as the periodic server model [20] or the minimum-parallelism supply model [21]. Here, a local scheduler determines the order in which jobs of an application are ordered, while a global scheduler determines the scheduling order of the reserved servers. The job queuing and processing policies may follow a single-queue or distributed multi-queue approach.

On-Demand Multi-Server Systems: This configuration models computer systems in cloud-computing in which jobs are assigned to on-demand servers and a fixed set of servers. Here, on-demand servers refer to the computational resources that can be accessed by an application almost instantaneously and returned to a server pool after the application completes it processing. The usage of on-demand servers may vary in terms of the number of servers and the duration for which they are available. This model can be applied to all previously discussed configurations.

2.3 Online Scheduling Variants

As the work in this thesis focuses on online scheduling, this section briefly describes the different variants of the problem depending on the job processing time information available at the time of decision-making.

Clairvoyant Online Scheduling: In the clairvoyant version of the online problem, the processing time of a job is made available to the scheduler upon arrival. In this setting, the Shortest-Remaining-Processing-Time (SRPT) algorithm is optimal for minimizing mean response times on single machines[18]. Algorithms such as SRPT are not common in practice since obtaining precise processing time for individual jobs is generally difficult. Yet, this setting provides valuable insights into optimal online scheduling.

Semi-Clairvoyant Online Scheduling: In the semi-clairvoyant version of the online problem, the *class* of a job is made available to the scheduler upon arrival instead of its exact processing time. A job's *class* is defined as an integer i such that the actual processing time of the job is in the range $[2^i, 2^{i+1}]$. The Lowest-Class-First (LCF) algorithm is known to be optimal in this setting for the objective of minimizing mean response times [22].

Stochastic Online Scheduling: In the stochastic version of the online problem, it is assumed that the information about the distribution of a job's processing time is known to the scheduler on the arrival of the job, but the true processing time is known only at its completion time [11]. The *MinIncrease* algorithm is known to be optimal for minimizing mean response times in this setting [23].

Prediction Clairvoyant Online Scheduling: In the prediction clairvoyant version of the online problem, it is assumed that the predicted value (or a proxy) of a job's processing time is available to the scheduler on the arrival of a job. The scheduler can use this information to decide the order in which the jobs are processed. It has been shown that even single-bit classifiers that distinguish long jobs from short jobs can significantly improve performance for minimizing mean response times [24].

Non-clairvoyant Online Scheduling: In the non-clairvoyant version of the online problem, it is assumed that no information about the processing time is available for an incoming job. A scheduler has to decide the order in which

incoming jobs are processed without knowing how long they will take to execute. For single-server settings, the round-robin algorithm has been shown to be 4-competitive [25].

2.4 Performance Objectives

The performance objectives of the work in the thesis are related to real-time guarantees. In real-time theory, a widely considered performance objective is that of satisfying *hard* real-time guarantees. In this scenario, jobs must access servers such that each job completes its execution before its deadline. Even one job missing its deadline results in failure. Another performance objective relates to providing *soft* real-time guarantees. This metric relaxes the condition that all jobs must complete their execution within their deadlines to one in which some jobs can miss their deadlines. Additionally, all jobs must complete their execution even if they have missed their deadlines. A third type of performance objective relates to providing *firm* real-time guarantees. This objective not only allows some jobs to miss their deadlines but also requires jobs that missed their deadlines to be removed from the queue of jobs that need access to servers.

The contributions of this thesis address each of the objectives outlined above. To further facilitate a comprehensive understanding of these performance objectives, the following definitions are essential:

Definition 1. *Tardiness (T_j) is defined as the maximum of zero or the difference between the completion time and the deadline of a job j, i.e.,*

$$T_j = \max(C_j - d_j, 0)$$

Definition 2. *Modified Tardiness ($\tilde{T_j}$) is defined as the maximum between completion time and deadline of a job j. i.e.,*

$$\tilde{T_j} = \max(C_j, d_j)$$

Definition 3. *Throughput is defined is as the ratio of the number of jobs that finish within their deadlines and the total number of released jobs in a measurement interval.*

With these definitions in place, the performance objectives considered in this thesis can now be formally stated.

Zero Tardiness: For applications with hard real-time guarantees, the objective is to ensure that all jobs finish within their deadlines; that is, given an instance of jobs J, the number of tardy jobs should be equal to zero, i.e., $\sum(U_j) = 0$, where U_j is defined as:

$$U_j = \begin{cases} 1 & \text{if } C_j > d_j, \\ 0 & \text{otherwise} \end{cases}$$

where C_j is the completion time of job j and d_j is its deadline.

Average Throughput Maximization: For applications with firm real-time guarantees, the objective is maximizing the average number of jobs that finish within their deadlines, given job arrival and processing time distributions. This can be expressed as:

$$\mathbf{E}\left[\sum_{j \in J}(1 - U_j)\right]$$

Total Modified Tardiness Minimization: Another tardiness-related objective for applications with soft real-time guarantees is minimizing the total modified tardiness of jobs. This can be expressed as:

$$\min \sum_{j \in J} \tilde{T}_j$$

Chapter 3

Related Work

Since the work in this thesis focuses on challenges related to scheduling with real-time requirements on multi-server systems, this chapter summarizes the current state-of-art closely related to the contributions in this thesis. This includes work related to ensuring zero tardiness in hierarchical multi-server systems (Section. 3.1), maximizing throughput in multi-server systems (Section. 3.2), minimizing tardiness in multi-server systems (Section. 3.3) and results related to scheduling with predictions (Section 3.4).

3.1 Hierarchical Scheduling in Multi-Server Systems

In a hierarchical scheduling framework, the computational needs of an application are abstracted by a single interface that specifies the computational time to be reserved along with the time period in which it should be provisioned [19]. The reserved computing time can be made available to the application through various reservation servers such as the periodic server and the deferrable server [20]. The mechanisms for defining such an interface and the schedulability tests vary depending on the scheduling strategies used for both local and global schedulers.

For single-server systems, Davis and Burns [20] provided an exact schedulability test for a hierarchical system with Fixed-Priority preemptive schedulers (FP) for local and global scheduling. They evaluated the schedulability under

periodic servers, sporadic servers, and deferrable servers. They concluded that the periodic servers provide better schedulability than the deferrable and sporadic servers. Similarly, Zhang and Burns [26] provided schedulability analysis for a system with the Earliest-Deadline-First (EDF) policy as the local scheduling policy and either FP or EDF as the global scheduling policy. Mok and Alex [27] proposed the regularity-based resource supply model and provided schedulability conditions for applications where either FP or EDF was used as the local scheduler. Shin and Lee [28] proposed the periodic resource supply model to define the guaranteed resource time for an application and the schedulability tests when FP or EDF is used as the local scheduling policy. Easwaran et al. [29] generalized the periodic resource model and provided methods to generate the optimal bandwidth interfaces and improve resource utilization. Dewan and Fisher [30, 31] proposed an algorithm to determine the optimal server parameters for the periodic resource model while Kim et al. [32] proposed a method to reduce the overhead associated with the periodic resource model. Yang and Dong [33] considered scheduling mixed-criticality tasks within a periodic resource model where each resource supply reservation had multiple bandwidth estimates.

For multi-server reservations, Leontyev and Anderson [34] proposed the minimum-parallelism supply model to schedule applications with soft real-time requirements. Yang and Anderson [21] provided conditions to preserve the optimality of the minimum-parallelism supply model for hard real-time applications. Easwaran et al. [35] extended the periodic resource model with an additional parameter that specifies the maximum number of physical servers that can be used to supply the reserved computation time at a given time. They investigated the schedulability of sporadic tasks within such a reservation using the global EDF scheduling policy [36]. In addition, they provided a transformation to generate equivalent periodic tasks for multi-server resource reservations (MPR) to be scheduled at the system level. Burmaykov et al. [37] proposed a generalization of the MPR model to reduce bandwidth allocation pessimism and provided schedulability conditions for global EDF and global FP scheduling policies. Pathan et al. [38] proposed an overhead-aware interface generation method for multi-server reservations with global FP scheduling policies.

Khalilzad et al. [39, 40, 41] proposed a feedback-based adaptive resource

reservation scheme that adjusts the bandwidth of resource supply for variable tasks to minimize deadline overruns. Groesbrink et al. [42] considered a similar approach to variable task management in which bandwidth is adjusted so that each server receives a guaranteed minimum supply while the remaining spare capacity is used to meet the demands of applications whose bandwidth should be increased. Cucinotta et al. [43] provided an adaptive scheme similar to the work presented in this thesis for applications with a finite set of modes and also consider power consumption as a constraint.

3.2 Throughput Maximization in Multi-Server Systems

Centralized Single-Queue Multi-Server Systems: When the number of machines is fixed, the problem of online throughput maximization has been extensively studied, especially in the clairvoyant and stochastic settings for both single and multiple servers. A well-known result is the optimality of the EDF algorithm for a set of feasible jobs on a single machine. Here, optimality refers to the fact that given any feasible set of jobs schedulable on a single machine, EDF schedules the jobs without missing any deadlines. For centralized single-queue multi-server systems where all jobs have unit weights, Phillips et al. [44] showed that EDF and Least-Laxity-First (LLF) are $(2 - \frac{1}{m})$-speed algorithms where m is the number of machines. If the objective is zero tardiness and extra machines are allowed compared to an offline algorithm, they showed that EDF needs at most $O(P)$ extra machines, where P is the ratio of the maximum and minimum processing times of the jobs in the instance whereas LLF needs at most $O(\log P)$ extra machines. Chen et al. [45] improved this and provided an $O(\log m)$-optimal algorithm. Azar [46] provided an improved algorithm that requires $O(\frac{\log m}{\log \log m})$ extra machines. Im et al. [47] further improved this result and provided an algorithm that needs $O(\log \log m)$ extra machines.

In firm real-time settings (i.e., when some jobs can be abandoned), Kalyanasundaram and Pruhs [48] provided a randomized algorithm that is O(1)-competitive for the problem of throughput maximization. For instances where all jobs have some slackness ϵ, Lucier et al. [49] provide $O(1 + \epsilon)$-speed, $O(\frac{1}{\epsilon})$-competitive algorithm. Moseley et al. [50] remove the require-

ment of slackness and provide the best known O(1)-competitive algorithm for this problem.

For a more general version of the problem where each job has a weight, Canetti and Irani [51] showed a $\Omega\left(\sqrt{\frac{\log k}{\log \log k}}\right)$ lower bound for any randomized algorithm where k is the minimum between the maximum job value, the maximum job duration, and the ratio between the maximum and minimum job value-density. Azar [52] showed an improved lower bound of $\Omega(\log k)$. Eberle [53] provided a $(\frac{1}{\epsilon})$-competitive algorithm for the case where jobs have slackness.

For the stochastic version of the problem, Abhaya et al. [54] provided a method to calculate the mean delay for the M/G/1/ queuing systems. Kargahi et al. [55] provided bounds for deadline miss probabilities for the M/M/k systems. Bryant et al. [56] showed that an EDF-based policy which postpones execution of certain jobs was optimal for scheduling on related machines when the total system load was less than one and jobs have exponentially distributed processing times.

Distributed Multi-Queue Multi-Server Systems: For distributed multi-queue multi-server systems, Kargahi et al. [57] considered stochastic online scheduling settings where jobs at individual servers are ordered according to the EDF policy. These jobs are assumed to be exponentially distributed with firm real-time requirements. They provided an approach to estimate deadline miss probabilities. As case studies, they considered three routing policies: Join-Shortest-Queue (JSQ), Minimal Expected Unfinished Work (MED), and a Threshold-Based Policy (TBP). JSQ assigns incoming jobs to the server with the least number of pending jobs. MED is a generalization of JSQ for heterogeneous servers, which assigns incoming jobs to the server with minimal expected unfinished work. With TBP, the assignment probabilities depend on a threshold for the number of pending jobs at individual servers.

3.3 Tardiness Minimization in Multi-Server Systems

Centralized Single-Queue Multi-Server Systems: The problem of minimizing tardiness even in offline settings and on a single machine is known to be

NP-hard [58]. Due to inherent difficulty in developing competitive algorithms for this problem a modified tardiness objective of the form $\sum w_j(T_j + d_j)$ was considered. This modified objective was introduced in the offline version of the problem with unit weights in which all jobs have a common deadline by Kovalyov and Werner [59]. Kolliopoulos and Steiner [60] considered the general version of the problem and showed a reduction to the problem of finding an approximate solution to the problem of weighted total completion time. Their result showed that any ρ-approximation algorithm for the problem of minimizing total weighted completion time was an $(\rho+1)$-approximation algorithm for the problem of minimizing weighted modified total tardiness. Liu et al. [61] extended this idea to online settings in addition to an availability constraint. They provided $O(1)$-competitive algorithms for clairvoyant scenarios. Specifically, for the single-machine version of the problem with weights, they showed a 2-competitive lower bound and a 3-competitive algorithm as an upper bound. For the multiple-machine version of the unit weight problem, they provided a lower bound of 1.309 and a 3-competitive algorithm. For the distributed multi-queue multi-server version of the problem that takes into account on-demand servers, Nakahira et al. [62] provided algorithms that minimized the mean and variance of the service capacity while minimizing the tardiness costs. In each of these scenarios, it is assumed that the knowledge of the processing time of a job is available at its release time. For the stochastic version of the problem, where the processing times are exponentially distributed, Bryant et al. [56] showed that the earliest-deadline-first was optimal for scheduling on multiple machines when the total system load was less than one.

Distributed Multi-Queue Multi-Server Systems: In multi-server environments, dispatching policies determine the server on which an incoming job will be executed. In online dispatching, the dispatching decision is made when the job arrives at a dispatching server. Dispatching policies, such as JSQ, and variants, such as Join-Shortest-Queue among k randomly chosen servers (JSQ-k), need to know the number of pending jobs in each of the considered servers [18]. Gathering this information may take significant time, depending on the number of servers and the network traffic. As an alternative, policies such as round-robin are agnostic to pending jobs on the servers and dispatch incoming jobs to

the servers in a repeating pattern. The advantage of policies such as round-robin is that they do not have the overhead associated with policies that require information about the pending workload. In most existing work, the objective of dispatching policies has been to minimize mean flow time. Several additional policies, such as join-the-idle-queue and join-below-threshold, where servers notify the dispatchers when they are idle or have pending jobs less than a predefined value, have been proposed to balance the trade-off between overheads and flow time [63, 64].

Hyytiä and Righter [65] considered the problem of routing jobs with known processing times and deadlines in distributed multi-queue multi-server systems. They consider the case where all jobs have a single waiting time deadline, and each server schedules jobs based on their arrival order. Additionally, jobs that miss their deadlines still need to complete. They developed a deadline-aware dispatching policy that outperforms the Least-Work-Left (LWL) dispatching policy concerning deadline violation probability for low-variance job processing time distributions and high-variance distributions. The dispatching policy assigns jobs to servers for which the expected weighted tardiness is the least. Hyytiä et al. [66] consider a slightly different problem in which all jobs have the same processing time requirements, but each job has a unique waiting time deadline. They developed a dispatching policy that relies on the first policy iteration of the random dispatch policy and showed that this policy performs best to minimize expected weighted tardiness. They further considered a generalized version of this problem in which job processing times are discrete. That is, the job processing times are random multiples of some concrete processing time value [67]. Here, too, their policy outperforms JSQ and LWL dispatching policies. Wang et al. [68] provided a load balancing and a core allocation strategy to minimize mean flow time on heterogeneous servers with both reserved and on-demand servers. Here, the on-demand servers are utilized when the queue is full or the waiting time exceeds a predefined maximum waiting time threshold.

3.4 Scheduling with Predictions

Much of the discussed related work assumes clairvoyant settings in which the job processing times are known or assume knowledge of their distributions.

In several practical scenarios, such information is hard to come by. To address this, non-clairvoyant algorithms in deterministic and randomized settings have been developed to minimize completion time and flow time objectives. Motwani et al. [14] presented a $\Omega(P)$-competitive deterministic algorithm to minimize total flow time on multiple servers. Bechhetti and Leonardi [69] developed a randomized version of the multi-level feedback algorithm which is $O(min\{\log n, \log \frac{n}{m}, \log n \log P\})$-competitive (Given n jobs and m machines). For semi-clairvoyant settings, they developed a $O(\log P)$-competitive algorithm. Several works have investigated the possibility of improving the performance of algorithms with machine-learned advice or predictions, including classical algorithms targeting problems such as online scheduling and load-balancing [70]. The evaluation of these prediction-augmented algorithms has been done in terms of competitive analysis under accurate predictions and for possibly incorrect predictions [24, 64, 71, 72]. Some algorithms rely on predicting job processing times [64, 73] while others rely on predicting job orders [72].

Since predictions are most likely to be erroneous, Scully et al. [73] showed that in an $M/G/1$ system, the predicted shortest job first algorithm had an approximation ratio bounded by the parameters α, β when estimated processing time of a job j with actual processing time p_j was within an interval $[\alpha p_j, \beta p_j]$ and Akbari et al. [74] developed a simple dynamic priority scheme that relies on estimated job processing times and showed that their algorithm performed quite well for the same objective of minimizing mean flow time for low variance prediction errors. Mitzenmacher [24] studied the impact of single-bit predictors that can indicate if a job's processing time is above or below some threshold in the context of large-scale queuing systems for the performance metric of mean flow time and showed that such predictors can provide benefits similar to those achievable with the knowledge exact processing times for Poisson arrivals and certain processing time distributions. The analysis identified the impact of incorrect predictions and highlighted the improvements achieved even with such incorrect predictions against an algorithm that chooses a queue with the least number of pending jobs. Similarly, they extended their analysis for the case where the individual job processing times are predicted and showed via simulations that the benefits of the *supermarket model* in large distributed systems were retained if the predictions were *reasonably precise* [71]. Based on the

evaluations, they recommended choosing the queue with least number of pending jobs and using the preemptive shortest predicted job first policy for ordering jobs on a single server for use in actual systems, as this combination performed well in a diverse range of scenarios.

Zhao et al. [75] extended the Randomized Multi-Level Feedback algorithm (RMLF) to use predicted job processing times to minimize mean flow time. Their experimental evaluation shows that the prediction-based algorithm achieves performance close to SRPT's when the prediction error is small. If the error is large, their algorithm can achieve performance no worse than RMLF. Azar et al. [76] developed a deterministic, non-migratory $O(\mu \log P)$-competitive algorithm for multiple servers. Unlike the algorithm developed by Zhao et al. [75], their algorithm requires no knowledge of the distortion parameter μ, which is the product of the ratio of maximum overestimation and maximum underestimation of true processing times. The work in this thesis is inspired by the findings of these studies and extends these approaches to objectives involving deadlines.

Chapter 4

Research Challenges

Scheduling algorithms have been extensively used to address the problem of managing resource access and the associated performance objectives in various settings. Nonetheless, architectural paradigms such as edge and fog computing, and the demands of emerging applications have created new challenges that require improved techniques to address these demands. This chapter provides an overview of the research challenges addressed in this thesis. The challenge related to ensuring zero tardiness is discussed in Section 4.1. The challenges related to throughput maximization and tardiness minimization are discussed in Section 4.2 and Section 4.3, respectively.

4.1 RC1: Zero Tardiness and Dynamic Demands

There exist classes of real-time applications where ensuring zero tardiness is of paramount importance. Key factors that facilitate this include approaches that identify and specify the amount of processing time required, the time intervals during which the processing time should be available, and implementations that provide this guaranteed amount of processing time in required intervals. A common practice to model such resource demand for some applications is to use the sporadic task model that specifies the demand of a single task τ_j using a triplet of the form $\{C_j, T_j, D_j\}$, where each element represents the worst-case processing time, the minimum inter-arrival time between consecutive requests

and the time before which the requested processing time should have been allocated, respectively. An application Γ then consists of a set of n such tasks $\{\tau_1, \cdots, \tau_n\}$. Using this information, one can analyze the schedulability of the application for algorithms such as earliest-deadline-first using associated schedulability tests [77].

Dynamic Computational Demand: For some systems such as industrial and mobile robots, the computational demand is influenced by the environment in which they operate. In static environments, it may be possible to precisely measure the demand, while in dynamic environments, the measurement might only be approximate. This demand variability can be observed both in the processing times and in the periodicity of the tasks [78]. For example, processing time variability can be due to different conditional branches in the code triggered by external events, while period variability can result from different control laws. Mode-based analysis [79, 80, 81] is an alternative approach to ensure the schedulability of applications with varying demands. In this approach, each mode is be defined by a specific set of tasks. When tasks exist in multiple different modes, they are distinguished by different processing times and periods. Schedulability is then evaluated for each mode and the transition intervals. In cases where such an approach is not possible, e.g., difficulty in identifying and defining different modes due to dynamic environments, or when schedulability within a mode cannot be guaranteed due to insufficient computational resources, it may be necessary to introduce some form of overload management to maintain the schedulability of the application [82, 83].

Elastic Task Model: The elastic task model [82, 84] provides an approach for specifying dynamic demands by allowing task parameters to be defined as a range of values rather than a single value as in the sporadic task model. It takes into account the possibility that task parameters can be changed at runtime to any value within the predefined range. An additional parameter, the elastic coefficient, is associated with each task to indicate the relative flexibility of the task to changes in its timing parameters. At runtime, overload management algorithms attempt to keep the application schedulable by modifying the task parameters to be as close as possible to their desired values based on the elastic coefficient of individual tasks.

RQ 1: What strategies can reduce the frequency of reservation bandwidth changes while ensuring zero tardiness for real-time applications with dynamic computational demands?

Reservation-Based Scheduling: Reservation-based scheduling approaches [85, 86] allocate processing time for each task according to its reservation parameters and schedule reservation servers rather than task instances (jobs). They allocate a fixed amount of computation time (called a budget) to a task during certain time intervals (called server periods) to ensure that each job of a task is completed by its deadline. Reservation servers allow each task to run for the duration of its budget within a server period. If a task has not completed its execution at the end of its budget, it is blocked until the next server period. If a task is released before the end of its minimum inter-arrival time, it will not be processed until its next server period starts. This approach ensures that tasks that violate their timing specifications do not affect the temporal behavior of other tasks in the system [83]. Static allocation of budget and the period, however, might be unable to meet the performance needs of applications with dynamic computational demands [87]. Applications with multiple modes and varying computational demands are managed by changing resource reservations and ensuring their schedulability during transition intervals [81, 41, 88]. In cases where sufficient spare capacity is available, the bandwidth changes can be made without affecting the performance of concurrently running applications or, alternatively, by considering their quality-of-service requirements [42]. An alternative mechanism to manage varying computational demand is to adjust the timing parameters of an application without changing the reservation parameters. Hierarchical adaptive reservation systems have been proposed for applications with distinct modes in [43, 89]. These solutions aim to change the reservation parameters so that all applications sharing the resource can run in optimal modes by considering the computational needs of all applications. This thesis extends this idea to support applications that do not have distinct modes but are rather specified according to the elastic task model to enable the execution of such applications in shared multi-server systems. Concretely, the work in this thesis addresses the research question RQ 1.

4.2 RC2: Throughput Maximization and Dynamic Capacity

Zero tardiness objective is the norm for control algorithms that manage critical components of a real-time system, such as the emergency stop functionality for industrial robots. Other components with dynamic demands may be more robust to occasional dropped jobs and perhaps even benefit if newer job instances based on fresh inputs are processed over older instances based on possibly stale inputs, thus allowing incomplete job instances to be dropped to process new jobs. Since ensuring zero tardiness often relies on resource allocation based on worst-case demands that may occur with low probability in some scenarios, it may result in underutilization of these resources. Instead of allocating resources based on worst-case demands, allocating resources based on average demands may provide improved resource utilization at a low or moderate loss in performance. Additionally, new architectural paradigms such as serverless computing and microservices are attractive due to their scaling capabilities thanks to the almost instantaneous on-demand availability of computational resources in large data centers and possibly in smaller geographically closer distributed data centers. A possible challenge is to consider how to effectively utilize these on-demand servers, given that accessing these servers dynamically may incur some costs. Therefore, the second challenge in the thesis deals with the objective of throughput maximization for firm real-time applications with on-demand availability of servers.

Focus on Clairvoyance: Substantial effort has been made to address the problem of throughput maximization for settings in which the processing times of jobs are precisely known or have known distributions. For instance, when the number of servers is fixed, randomized and deterministic algorithms exist that are $O(1)$-competitive [48, 50]. For the general model of weighted throughput on heterogeneous servers with slackness ϵ, there is a $O(\frac{1}{\epsilon})$- competitive algorithm [53]. These algorithms rely on precise knowledge of a job's processing times on its arrival. For the stochastic model where processing times are exponentially distributed, a migratory EDF-based algorithm has been shown to be optimal with respect to average throughput if the system load is less than one

on related servers [56].

Dynamic Service Capacity: Very few results consider the on-demand availability of service capacity for the throughput maximization problem. The only known result appears to be the generalized exact distributed scheduler [62] to minimize the variance of service capacity for jobs with deadlines. A closely related setting appears to be the dynamic bin-packing problem where the objective is to minimize the usage time of bins when jobs occupy the bins for a fixed duration of time and depart once they reach their deadlines [90, 91, 92]. This can be considered as a distributed multi-queue multi-server system where each server schedules jobs in their arrival order and processes them non-preemptively. Jobs are assigned to already open servers only if they can be processed on that server before their deadlines, otherwise they are dispatched to an on-demand server. Unfortunately, these approaches rely on knowledge of job processing times. Therefore, the work in this thesis addresses the research question RQ 2.

> **RQ 2:** Which approaches can improve average throughput for firm real-time applications with limited job processing time information in on-demand distributed multi-queue multi-server systems?

4.3 RC3: Tardiness and Non-Clairvoyance

Another challenge that is addressed in the thesis relates to scenarios where a large amount of data that acts as input to control algorithms needs to be processed. For instance, controlling a mobile robot using a vision-based motion planning algorithm deployed on the cloud may require transmitting images from multiple sources to localize the robot's position and plan an obstacle-free path for it to navigate. In this scenario, it may be beneficial from an end-to-end perspective to continue running the request to completion even if the job is past its deadline. The delay due to re-transmission and re-computation may not be worthwhile. Therefore, the third challenge addressed in this thesis relates to the problem of tardiness minimization in multi-server systems.

RQ 3: What strategies can be employed to minimize modified total tardiness for soft real-time applications with limited job processing time information in centralized single-queue multi-server systems?

Focus on Clairvoyance: Additionally, several variants of the tardiness minimization problem have been investigated as discussed in Section 3.3. However, no optimal algorithm is known even in clairvoyant settings. To overcome the difficulty in developing competitive algorithms for this objective [60], a modified tardiness objective has been considered. For this modified objective, the best-known results appear to be a 3-competitive algorithm by Liu et al. [61]. The dispatching policies by Hyytiä et al. [65, 66, 67] perform the best for the original tardiness objective in distributed multi-queue multi-server systems. When on-demand servers are allowed, Nakahari et al. [62] provide an efficient algorithm for the considered settings. These approaches, however, depend on precise knowledge of the processing times of jobs. While no known result exists for the non-clairvoyant scenario, following the reduction approach in [60, 61], it is straightforward to show a $O(P)$-competitive upper bound for the non-preemptive run-to-completion algorithm by Motwani et al. [14]. Therefore, the third challenge addresses the research question RQ 3.

Chapter 5

Research Methodology

The thesis results were developed following the hypothetico-deductive method [93]. This method involves making hypotheses based on existing theories and then deducing the logical consequences of these hypotheses. The research process consisted of the following four steps:

- **Problem Definition** - This step involves understanding the field of the topic through a comprehensive literature review, state-of-art and state-of-practice studies to identify gaps. Based on this, the scope of the problem is defined, setting the boundaries for the research. As an outcome of this approach, the three research challenges were identified.

- **Idea Development** - This step involves the iterative development of solutions to the defined problem. It starts with brainstorming and ideation, as well as proposing and discussing various potential solutions. These ideas are then refined and developed further, considering the problem's constraints and requirements. The solutions in this thesis addressing the identified research challenges were designed following this approach.

- **Implementation** - This step involves converting the idea and theory into an artifact. The proposed solution is implemented, often through a combination of coding, experimentation, and prototyping. This step is crucial for testing the solution's performance and identifying any practical

Research Methods	**Research Question**
State-of-Art Study	RQ 1, RQ 2, RQ 3
State-of-Practice Study	RQ 1, RQ 2, RQ 3
Simulation	RQ 1, RQ 2
Algorithm Design and Analysis	RQ 3

Table 5.1: Mapping of research methods utilized to address research questions in this thesis.

issues that might arise. The algorithms presented in this thesis were implemented and evaluated using C++ programs.

- **Evaluation** - This step involves evaluating the idea and its implementation. The solution is tested against predefined criteria or benchmarks to assess its effectiveness. The results of these tests are then analyzed and interpreted to draw conclusions about the solution's performance. This step involves comparing other existing solutions to determine the relative strengths and weaknesses of the proposed solution. While most solutions in this thesis were evaluated using simulations, a few were evaluated using competitive analysis.

Chapter 6

Thesis Contributions

Building on the research challenges and questions outlined in the previous chapter, this chapter presents the thesis contributions that address scheduling in multi-server systems and the performance requirements of real-time applications. Section. 6.1 summarizes the contribution addressing the research question related to the zero tardiness objective. Section. 6.2 summarizes the contribution addressing research question related to average throughput improvement. Section. 6.3 summarizes the contribution addressing the research question related to tardiness minimization.

Table. 6.1 establishes the relationship between the included papers and the research questions formulated in Chapter 4.

6.1 C1: Minimizing Bandwidth Changes in Hierarchical Multi-Server Systems

Topic: This contribution addresses the research question **RQ 1** and forms the main content of Paper A. It addresses the challenges of resource allocation and scheduling in hierarchical multi-server systems where applications may have computational needs that vary over time. This work bridges the gap between traditional real-time scheduling theory and the demands of adaptive computing environments.

Contribution	RQ 1	RQ 2	RQ 3
C1	Paper A	-	-
C2	-	Paper B,C and D	-
C3	-	-	Paper E

Table 6.1: Mapping of Research Contributions and Research Questions

Goal: This contribution focuses on minimizing reservation bandwidth changes while maintaining zero tardiness in hierarchical multi-server systems where several independent real-time applications can share the servers. This goal has been defined to ensure that other independent applications running on the same hardware are minimally impacted. To achieve this goal, a scheduling approach has been developed that minimizes reservation bandwidth changes for elastic tasks while satisfying period change requests. The approach combines per-server utilization modification, task re-partitioning, and reservation bandwidth adjustment.

Approach: This work considers a system model where elastic tasks must execute within reservation servers designed according to the multiprocessor minimum-parallelism resource supply form [21, 34]. This reservation approach offers applications a fixed number of fully dedicated servers and, at most, one partial server, available according to the periodic resource supply model. Each reservation has a local scheduler that manages application tasks. This scheduler adheres to the partitioned EDF scheduling policy, with reservation parameters initially set based on the desired utilization and period of the application's tasks. The primary goal is to minimize the frequency of reservation bandwidth changes once the initial parameters are established. To achieve this, when a task requests a period change, the utilization modification algorithm focuses solely on the tasks sharing the server with the requesting task. It attempts to generate new periods that satisfy the constraints of these affected tasks. If the algorithm fails to find a solution, a two-step approach is employed:

First, an attempt is made to re-partition the tasks among the servers, including the partial server, using a reasonable allocation scheme [94]. If re-partitioning is unsuccessful, the reservation bandwidth is then considered for

modification. This modification may involve adjusting the partial server bandwidth or, if necessary, allocating a new fully dedicated server.

The effectiveness of this approach is demonstrated through an evaluation presented in paper A. The per-server utilization modification scheme can satisfy up to 94 percent of requests. When combined with the re-partitioning step, a 100 percent success rate for period change requests is achieved, highlighting the approach's robustness in managing dynamic task requirements while considering the impact on other co-located applications.

Potential Impact: This contribution has implications for practical applications and academic research. In practice, this approach can help ensure schedulability by adjusting the application demands when additional capacity is unavailable. From an academic perspective, this study provides a basis for further research on elastic task scheduling in compositional systems. It offers a reference point for comparing future adaptive scheduling algorithms.

Limitations: While this contribution remains valuable, some aspects could be addressed in future work. An area for potential improvement could be examining overhead costs associated with frequent task re-partitioning, which the current work does not explicitly address. Additionally, the approach currently focuses on partitioned EDF as the local scheduling policy, which may limit its applicability in systems using different scheduling algorithms. Another area of improvement could be the run-time complexity of the approach. While the run-time complexity of this approach's task period assignment algorithm is quadratic in the number of tasks, this design provides only a probabilistic and slight improvement by reducing the number of tasks that may need to be modified. A more efficient approach may be desirable.

6.2 C2: Improving Average Throughput in Distributed Multi-Server Systems

Topic: This contribution addresses the research question **RQ 2** and is provided in papers B, C, and D. The broader area of contribution of these papers is

real-time scheduling and load balancing in on-demand multi-server systems. It focuses on managing jobs with stochastic execution times, inter-arrival times, and deadlines in multi-server systems, emphasizing meeting firm real-time requirements without precise processing time information. This work bridges the gap between traditional real-time systems and emerging edge computing paradigms, addressing the challenges due to on-demand resource availability.

Goal: The goal of this research is to develop online algorithms to maximize average throughput for a distributed multi-queue multi-server system with on-demand resource availability when inputs arrive following a Poisson process; the jobs processing times are drawn from an exponential distribution. The studies compare prediction-based admission and scheduling policies against traditional approaches like EDF, assessing their performance in meeting deadlines. A key objective has been to investigate the effectiveness of on-demand resource allocation for satisfying firm real-time requirements, particularly in scenarios where job processing times are not fully known in advance.

Approach: In paper B, the approach involves considering a dual-bit job size predictor that classifies an incoming job into one of four job size classes: small, medium, large, and very large. This information is then used with a Preemptive Shortest-Job-Class-First (PSJF) scheduling policy. The policy orders jobs according to their job size classes, and jobs in each class are in the First-In-First-Out (FIFO) order. The evaluation indicates that FIFO and EDF policies outperform the prediction-based policy in under-loaded conditions. However, in overloaded scenarios, the prediction-based policy offers better performance.

In paper C, the framework integrates an on-arrival dispatcher with EDF scheduling for an on-demand distributed multi-queue multi-server system. The dispatching policy uses a schedulability test that estimates job response times based on the number of pending jobs in each queue and dispatches jobs to servers likely to meet their deadlines. The dispatcher employs a Join-Shortest-Queue (JSQ) policy for assigning jobs across the servers. Simulations demonstrate that this approach significantly enhances throughput. It outperforms the first-fit heuristic and systems using only reserved servers, especially when jobs are dispatched to on-demand servers.

In paper D, a single-bit execution time prediction-based admission policy for online dispatching and scheduling jobs in distributed multi-queue multi-server systems is explored. The performance of this policy is compared against mean-approximation and clairvoyant policies, using simulations to evaluate throughput under various loads. The results indicate that the single-bit prediction policy outperforms the mean approximation policy but remains less effective than the clairvoyant policy which has precise processing time information of each of the jobs.

Potential Impact: This contribution has the potential for impact in both practical and academic settings. In practice, the results could lead to improved resource management in edge computing platforms for real-time applications, enhancing performance for industrial embedded systems that require offloading to edge resources. Academically, this work advances the understanding of real-time scheduling in distributed edge computing environments, providing new insights into the trade-offs between prediction-based and traditional scheduling policies for throughput maximization.

Limitations: The contributions rely primarily on simulation-based evaluation, which may not capture all real-world complexities. Additionally, the impact of network delays and communication overhead is not fully explored, potentially limiting the applicability of the findings in highly distributed environments.

6.3 C3: Minimizing Total Tardiness in Centralized Multi-Server Systems

Topic: This contribution addresses the research question **RQ 3** for soft real-time applications in centralized single-queue multi-server systems and is the main content of Paper E. The research explores using predictions to minimize total tardiness, addressing scheduling challenges in environments where job characteristics may not be fully known in advance. This work bridges the gap

between traditional real-time scheduling and prediction based approaches, aiming to improve system performance as measured by job tardiness.

Goal: The primary goal of this research has been to develop and evaluate prediction-based scheduling algorithms that can effectively minimize total tardiness in soft real-time applications. A key objective has been to investigate how incorporating predictions can enhance the outcome of scheduling decisions in centralized multi-server environments, particularly when dealing with uncertainties in job processing times and arrival patterns.

Approach: Three algorithms are proposed in paper E. All three algorithms were originally designed for minimizing flow times and completion times which are different from the tardiness minimization objective considered in this thesis. These algorithms prioritize jobs based on different criteria, such as job classes based on predicted processing time, actual class of the job, and job density calculated considering weight, predicted speeds, and known processing times. The first algorithm is $O(\mu \log P)$-competitive for scheduling unit weight jobs on parallel identical machines in prediction-clairvoyant settings. Similarly, the second algorithm is $O(\log P)$-competitive when scheduling unit weight jobs on parallel identical machines in semi-clairvoyant settings while the third algorithm is $O(\mu)$-competitive for scheduling arbitrary weight jobs on parallel unrelated machines and clairvoyant processing times in speed-prediction settings.

Potential Impact: Academically, this work advances the understanding of prediction-based scheduling in real-time systems, providing new insights into the trade-offs between prediction accuracy and scheduling performance for soft real-time jobs. It establishes a baseline for further research on integrating machine learning techniques with real-time scheduling.

Limitations: The focus on centralized single-queue multi-server systems limits the applicability of the findings to distributed or decentralized architectures. Moreover, this contribution does not consider the computational over-

head of making and utilizing predictions in real-time scenarios, which could impact overall system performance.

6.4 Personal Contributions

My personal contributions in the papers included in this thesis are summarized based on CRediT[1] (Contributor Roles Taxonomy) definitions in Table 6.2.

CRediT Contribution	Paper A	Paper B	Paper C	Paper D	Paper E
Conceptualization	✓	✓	✓	✓	✓
Data curation	✓	✓	✓	✓	✓
Formal analysis	✓				✓
Funding acquisition					
Investigation	✓	✓	✓	✓	✓
Methodology	✓	✓		✓	✓
Project administration					
Resources					
Software	✓	✓	✓	✓	✓
Supervision					
Validation	✓	✓	✓	✓	✓
Visualization	✓	✓	✓	✓	✓
Writing – original draft	✓	✓	✓	✓	✓
Writing – review and editing	✓	✓	✓	✓	✓

Table 6.2: Personal contribution following CRediT in included papers

6.5 Included Papers

Paper A

Title: Multi-processor scheduling of elastic applications in compositional real-time systems [95]
Authors: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte

[1] `https://credit.niso.org/`

Status: Published in Journal of Systems Architecture, (December 2021)
Abstract: Scheduling of real-time applications modeled according to the periodic and the sporadic task model under hierarchical and compositional real-time systems has been widely studied to provide temporal isolation among independent applications running on shared resources. However, for some real-time applications that are amenable to variation in their timing behavior, usage of these task models can result in pessimistic solutions. The elastic task model addresses this pessimism by allowing the timing requirements of an application's tasks to be specified as a range of values instead of a single value. Although the scheduling of elastic applications on dedicated resources has received considerable attention, there is limited work on scheduling such applications in hierarchical and compositional settings.

In this paper, we evaluate different earliest deadline first scheduling algorithms to schedule elastic applications in a minimum parallelism supply form reservation on a multiprocessor system. Our evaluation indicates that the proposed approach provides performance comparable to the current state-of-the-art algorithms for scheduling elastic applications on dedicated processors in terms of schedulability.

Paper B

Title: Evaluating Dispatching and Scheduling Strategies for Firm Real-Time Jobs in Edge Computing [96]
Authors: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte
Status: Published in the proceedings of 49th Annual Conference of the IEEE Industrial Electronics Society (IECON 2023)
Abstract: We consider the problem of on-arrival dispatching and scheduling jobs with stochastic execution times, inter-arrival times, and deadlines in multi-server fog and edge computing platforms. In terms of mean response times, it has been shown that size-based scheduling policies, when combined with dispatching policies such as join-shortest-queue, provide better performance over policies such as first-in-first-out. Since job sizes may not always be known apriori, prediction-based policies have performed reasonably well. However, little is known about the performance of prediction-based policies for jobs with firm

deadlines. In this paper, we address this issue by considering the number of jobs that complete within their deadlines as a performance metric and investigate, using simulations, the performance of a prediction-based shortest-job-first scheduling policy for the considered metric and compare it against scheduling policies that prioritize based on deadlines (EDF) and arrival times (FIFO). The evaluation indicates that in under-loaded conditions, the prediction-based policy is outperformed by both FIFO and EDF policies. However, in overloaded scenarios, the prediction-based policy offers slightly better performance.

Paper C

Title: Dispatching deadline-constrained jobs in edge computing systems [97]
Authors: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte
Status: Published in the proceedings of IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA 2023)
Abstract: The edge computing paradigm extends the architectural space of real-time systems by bringing the capabilities of the cloud to the edge. Unlike cloud-native systems designed for mean response times, real-time industrial embedded systems are designed to control a single physical system, such as a manipulator arm or a mobile robot, that requires temporal predictability. We consider the problem of dispatching and scheduling of jobs with deadlines that can be offloaded to the edge and propose DAL, a deadline-aware load balancing and scheduling framework that leverages the availability of on-demand computing resources along with an on-arrival dispatching scheme to manage temporal requirements of such offloaded applications. The evaluation indicates that DAL can achieve reasonably good performance even when execution times, arrival times, and deadlines vary.

Paper D

Title: Scheduling firm real-time applications on the edge with single-bit execution time prediction [98]
Authors: Shaik Mohammed Salman, Van Lan Dao, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte

Status: Published in the proceedings of IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC 2023)
Abstract: The edge computing paradigm brings the capabilities of the cloud, such as on-demand resource availability, to the edge for applications with low latency and real-time requirements. While cloud-native load balancing and scheduling algorithms strive to improve performance metrics like mean response times, real-time systems that govern physical systems must satisfy deadline requirements. This paper explores the potential of an edge-computing architecture that utilizes the on-demand availability of computational resources to satisfy firm real-time requirements for applications with stochastic execution and inter-arrival times. As it might be difficult to know the precise execution times of individual jobs before completion, we consider an admission policy that relies on single-bit execution time predictions for dispatching. We evaluate its performance in terms of the number of jobs that complete by their deadlines via simulations. The results indicate that the prediction-based admission policy can achieve reasonable performance for the considered settings.

Paper E

Title: Taming Tardiness on Parallel Machines: Online Scheduling with Limited Job Information [99]
Authors: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte
Status: MRTC Report. MDU 2024
Abstract: We consider the problem of scheduling n jobs with soft deadlines on M parallel machines in online settings. Since no bounded competitive algorithms exist to minimize total tardiness $\sum w_j T_j$, we consider an objective of the form $\sum w_j(T_j + d_j)$. For this modified objective, we provide competitive algorithms with different levels of information about job processing times.

Chapter 7

Conclusion

7.1 Conclusion

In this thesis, challenges in scheduling real-time applications in multi-server systems were addressed through the investigation of three key research questions. The first question investigated how the frequency of reservation bandwidth changes could be reduced while ensuring zero tardiness for real-time applications with dynamic computational demands, assuming a setting with complete knowledge of job processing times. The second question explored approaches to improve average throughput for firm real-time applications with limited job information in on-demand distributed multi-queue multi-server systems, given knowledge of job size distributions. Finally, the third question examined methods to minimize tardiness for soft real-time applications in centralized single-queue multi-server systems, specifically in non-clairvoyant settings where exact job processing times are unknown. For each of these research questions, solutions are proposed in the papers included in this thesis.

To address the first research question, a framework was developed to minimize reservation bandwidth changes in compositional multi-server systems while ensuring zero tardiness. For the second question, a framework was presented that aimed to improve average throughput for firm real-time applications in distributed multi-queue multi-server systems by utilizing processing time predictions. Finally, to address the third research question, a set of online

scheduling algorithms was introduced for soft real-time applications aimed at minimizing tardiness in centralized single-queue multi-server systems. These algorithms, characterized by their competitive ratios, utilized varying levels of information regarding processing times to achieve their objective.

7.2 Future Directions

Building on the insights derived from investigation of the research questions, some potential directions for future research can be identified. These directions could extend the findings presented and address further challenges in scheduling real-time applications within multi-server systems.

Zero Tardiness: Zero tardiness is a critical requirement in many real-time applications, particularly those that demand strict adherence to deadlines. One promising direction is to examine the effectiveness of the approach proposed in this thesis by employing local scheduling policies other than partitioned EDF. This investigation may provide valuable insights into how various scheduling algorithms interact with the elastic task model and reservation server framework. Additionally, considering an elastic parallel task model in a hierarchical multi-server system could represent a meaningful extension of the current work, contributing to a deeper understanding of task allocation and resource utilization in more complex environments.

Throughput Maximization: Maximizing throughput is essential for ensuring that systems can efficiently handle the workload of firm real-time applications, particularly in environments where resources are limited or variable. Another avenue worth exploring is the integration of machine learning techniques for enhancing the accuracy of execution time predictions, which could potentially improve the effectiveness of the proposed scheduling approaches. Developing theoretical bounds and analytical models for these strategies may offer a stronger foundation for understanding their behavior and limitations. Furthermore, extending the scope of this research to consider end-to-end latency in complex edge-cloud ecosystems could address broader system-level

integration challenges. Investigating the use of heterogeneous edge computing resources with varying processing capabilities may also yield insights into optimizing performance in more realistic hardware configurations.

Tardiness Minimization: Tardiness minimization is particularly important in soft real-time applications, where occasional deadline misses may be acceptable but should be kept to a minimum to maintain quality of service. The algorithms proposed in this thesis for minimizing tardiness primarily focus on the availability of a fixed number of servers. A potential direction for future work could involve integrating these scheduling algorithms with the concept of on-demand server availability, thereby offering more flexibility and responsiveness in dynamic environments. Additionally, exploring performance within a distributed multi-queue multi-server system could lead to scalable solutions and improved efficiency in resource allocation. Through the investigation of these areas, future research could build upon the findings of this thesis and contribute to the development of effective scheduling strategies for real-time applications.

Bibliography

[1] Hermann Kopetz. *The Real-Time Environment*, pages 1–28. Springer US, Boston, MA, 2011.

[2] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. *National institute of science and technology, special publication*, 800(2011):145, 2011.

[3] Michaela Iorga, Larry Feldman, Robert Barton, Michael J Martin, Nedim S Goren, Charif Mahmoudi, et al. Fog computing conceptual model. Technical report, National Institute of Standards and Technology, 2018.

[4] Shaik Mohammed Salman, Vaclav Struhar, Alessandro V Papadopoulos, Moris Behnam, and Thomas Nolte. Fogification of industrial robotic systems: Research challenges. In *Proceedings of the Workshop on Fog Computing and the IoT*, pages 41–45, 2019.

[5] Shaik Mohammed Salman, Taufik Akbar Sitompul, Alessandro Vittorio Papadopoulos, and Thomas Nolte. Fog computing for augmented reality: Trends, challenges and opportunities. In *2020 IEEE International Conference on Fog Computing (ICFC)*, pages 56–63. IEEE, 2020.

[6] Mohammed Salman Shaik, Václav Struhár, Zeinab Bakhshi, Van-Lan Dao, Nitin Desai, Alessandro V Papadopoulos, Thomas Nolte, Vasileios Karagiannis, Stefan Schulte, Alexandre Venito, et al. Enabling fog-based industrial robotics systems. In *2020 25th IEEE International Conference*

on Emerging Technologies and Factory Automation (ETFA), volume 1, pages 61–68. IEEE, 2020.

[7] Victor Millnert, Johan Eker, and Enrico Bini. Achieving predictable and low end-to-end latency for a network of smart services. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2018.

[8] Michael Pearce, Sherali Zeadally, and Ray Hunt. Virtualization: Issues, security threats, and solutions. *ACM Computing Surveys (CSUR)*, 45(2):1–39, 2013.

[9] Václav Struhár, Silviu S Craciunas, Mohammad Ashjaei, Moris Behnam, and Alessandro V Papadopoulos. Hierarchical resource orchestration framework for real-time containers. *ACM Transactions on Embedded Computing Systems*, 23(1):1–24, 2024.

[10] Mor Harchol-Balter. Open problems in queueing theory inspired by datacenter computing. *Queueing Systems*, 97(1):3–37, 2021.

[11] Michael L. Pinedo. Offline deterministic scheduling, stochastic scheduling, and online deterministic scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[12] Mohammadreza Barzegaran and Paul Pop. Communication scheduling for control performance in tsn-based fog computing platforms. *IEEE Access*, 9:50782–50797, 2021.

[13] Paul Pop, Konstantinos Alexandris, and Tongtong Wang. Configuration of multi-shaper time-sensitive networking for industrial applications. *IET Networks*, 2024.

[14] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994.

[15] Susanne Albers. Online scheduling. In *Introduction to scheduling*, pages 71–98. CRC Press, 2009.

[16] Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[17] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

[18] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[19] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3):30:1–30:39, 2008.

[20] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. *Proceedings - Real-Time Systems Symposium*, 2005.

[21] Kecheng Yang and James H. Anderson. On the Dominance of Minimum-Parallelism Multiprocessor Supply. *Proceedings - Real-Time Systems Symposium*, 0:215–226, 2016.

[22] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Semi-clairvoyant scheduling. *Theoretical computer science*, 324(2-3):325–335, 2004.

[23] Nicole Megow, Marc Uetz, and Tjark Vredeveld. Stochastic online scheduling on parallel machines. In *Approximation and Online Algorithms: Second International Workshop, WAOA 2004, Bergen, Norway, September 14-16, 2004, Revised Selected Papers 2*, pages 167–180. Springer, 2005.

[24] Michael Mitzenmacher. Queues with small advice. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, pages 1–12. SIAM, 2021.

[25] Benjamin Moseley and Shai Vardi. The efficiency-fairness balance of round robin scheduling. *Operations Research Letters*, 50(1):20–27, 2022.

[26] Fengxiang Zhang and Alan Burns. Analysis of hierarchical edf preemptive scheduling. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 423–434, 2007.

[27] Aloysius K Mok and Xiang Alex. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*, pages 129–138. IEEE, 2001.

[28] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2–13. IEEE Computer Society, 2003.

[29] Arvind Easwaran, Insup Lee, Insik Shin, and Oleg Sokolsky. Compositional schedulability analysis of hierarchical real-time systems. In *ISORC*, pages 274–281. IEEE Computer Society, 2007.

[30] Nathan Fisher and Farhana Dewan. A bandwidth allocation scheme for compositional real-time systems with periodic resources. *Real Time Syst.*, 48(3):223–263, 2012.

[31] Farhana Dewan and Nathan Fisher. Bandwidth allocation for fixed-priority-scheduled compositional real-time systems. *ACM Trans. Embed. Comput. Syst.*, 13(4):91:1–91:29, 2014.

[32] Jin Hyun Kim, Kyong Hoon Kim, Arvind Easwaran, and Insup Lee. Towards overhead-free interface theory for compositional hierarchical real-time systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 37(11):2869–2880, 2018.

[33] Kecheng Yang and Zheng Dong. Mixed-criticality scheduling in compositional real-time systems with multiple budget estimates. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 25–37. IEEE, 2020.

[34] Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Proceedings - Euromicro Conference on Real-Time Systems*, pages 191–200, 2008.

[35] Arvind Easwaran, Insup Lee, Oleg Sokolsky, and Steve Vestal. A compositional scheduling framework for digital avionics systems. In *RTCSA*, pages 371–380. IEEE Computer Society, 2009.

[36] Sanjoy Baruah, Marko Bertogna, and Giorgio C. Buttazzo. *Multiprocessor scheduling for real-time systems*. Embedded systems. Springer, Cham, 2015.

[37] Artem Burmyakov, Enrico Bini, and Eduardo Tovar. Compositional multiprocessor scheduling: the GMPR interface. *REAL-TIME SYSTEMS*, 50(3, SI):342–376, MAY 2014.

[38] Risat Mahmud Pathan, Per Stenström, Lars Göran Green, Torbjörn Hult, and Patrik Sandin. Overhead-aware temporal partitioning on multicore processors. *Real-Time Technology and Applications - Proceedings*, 2014-Octob(October):251–262, 2014.

[39] Nima Moghaddami Khalilzad, Moris Behnam, and Thomas Nolte. Adaptive hierarchical scheduling framework: Configuration and evaluation. In *ETFA*, pages 1–10. IEEE, 2013.

[40] Nima Moghaddami Khalilzad, Moris Behnam, and Thomas Nolte. Multilevel adaptive hierarchical scheduling framework for composing real-time systems. In *RTCSA*, pages 320–329. IEEE Computer Society, 2013.

[41] Nima Khalilzad, Fanxin Kong, Xue Liu, Moris Behnam, and Thomas Nolte. A feedback scheduling framework for component-based soft real-time systems. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 182–193, 2015.

[42] Stefan Groesbrink, Luis Almeida, Mario De Sousa, and Stefan M. Petters. Towards certifiable adaptive reservations for hypervisor-based virtualization. *Real-Time Technology and Applications - Proceedings*, 2014-October(October):13–24, 2014.

[43] Tommaso Cucinotta, Luigi Palopoli, Luca Abeni, Dario Faggioli, and Giuseppe Lipari. On the integration of application level and resource level

QoS control for real-time applications. *IEEE Transactions on Industrial Informatics*, 6(4):479–491, 2010.

[44] Cynthia A Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 140–149, 1997.

[45] Lin Chen, Nicole Megow, and Kevin Schewior. An o(m)-competitive algorithm for online machine minimization. *SIAM Journal on Computing*, 47(6):2057–2077, 2018.

[46] Yossi Azar and Sarel Cohen. An improved algorithm for online machine minimization. *Operations Research Letters*, 46(1):128–133, 2018.

[47] Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Clifford Stein. An o (log log m)-competitive algorithm for online machine minimization. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 343–350. IEEE, 2017.

[48] Bala Kalyanasundaram and Kirk R Pruhs. Maximizing job completions online. *Journal of Algorithms*, 49(1):63–85, 2003.

[49] Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 305–314, 2013.

[50] Benjamin Moseley, Kirk Pruhs, Clifford Stein, and Rudy Zhou. A competitive algorithm for throughput maximization on identical machines. *Mathematical Programming*, pages 1–18, 2024.

[51] Ran Canetti and Sandy Irani. Bounding the power of preemption in randomized scheduling. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 606–615, 1995.

[52] Yossi Azar and Oren Gilon. Scheduling with deadlines and buffer management with processing requirements. *Algorithmica*, 78:1246–1262, 2017.

[53] Franziska Eberle. O $(1/\varepsilon)$ is the answer in online weighted throughput maximization. In *41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[54] Vidura Gamini Abhaya, Zahir Tari, Panlop Zeephongsekul, and Albert Y. Zomaya. Performance analysis of edf scheduling in a multi-priority preemptive M/G/1 queue. *IEEE Transactions on Parallel and Distributed Systems*, 25(8):2149–2158, 2014.

[55] Mehdi Kargahi and Ali Movaghar. A method for performance analysis of earliest-deadline-first scheduling policy. *Journal of Supercomputing*, 37(2):197–222, 2006.

[56] Richard Bryant, Peter Lakner, and Michael Pinedo. On the optimality of the earliest due date rule in stochastic scheduling and in queueing. *European Journal of Operational Research*, 298:202–212, 4 2022.

[57] Mehdi Kargahi and Ali Movaghar. Dynamic routing of real-time jobs among parallel edf queues: A performance study. *Computers & Electrical Engineering*, 36(5):835–849, 2010.

[58] Richard K Congram, Chris N Potts, and Steef L van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

[59] Mikhail Y Kovalyov and Frank Werner. Approximation schemes for scheduling jobs with common due date on parallel machines to minimize total tardiness. *Journal of Heuristics*, 8:415–428, 2002.

[60] Stavros G Kolliopoulos and George Steiner. Approximation algorithms for scheduling problems with a modified total weighted tardiness objective. *Operations research letters*, 35(5):685–692, 2007.

[61] Ming Liu, Yinfeng Xu, Chengbin Chu, and Feifeng Zheng. Online scheduling to minimize modified total tardiness with an availability constraint. *Theoretical computer science*, 410(47-49):5039–5046, 2009.

[62] Yorie Nakahira, Andres Ferragut, and Adam Wierman. Generalized exact scheduling: A minimal-variance distributed deadline scheduler. *Operations Research*, 71(2):433–470, 2023.

[63] Xingyu Zhou, Fei Wu, Jian Tan, Yin Sun, and Ness Shroff. Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–30, 2017.

[64] Yecheng Zhao, Runzhi Zhou, and Haibo Zeng. Design optimization for real-time systems with sustainable schedulability analysis. *Real-Time Systems*, 58(3):275–312, 2022.

[65] Esa Hyytiä and Rhonda Righter. Routing jobs with deadlines to heterogeneous parallel servers. *Operations Research Letters*, 44(4):507–513, 2016.

[66] Esa Hyytiä, Rhonda Righter, Olivier Bilenne, and Xiaohu Wu. Dispatching fixed-sized jobs with multiple deadlines to parallel heterogeneous servers. *Performance Evaluation*, 114:32–44, 2017.

[67] Esa Hyytiä, Rhonda Righter, Olivier Bilenne, and Xiaohu Wu. Dispatching discrete-size jobs with multiple deadlines to parallel heterogeneous servers. *Systems modeling: methodologies and tools*, pages 29–46, 2019.

[68] Shuang Wang, Xiaoping Li, Quan Z Sheng, Ruben Ruiz, Jinquan Zhang, and Amin Beheshti. Multi-queue request scheduling for profit maximization in iaas clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32(11):2838–2851, 2021.

[69] Luca Becchetti and Stefano Leonardi. Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 94–103, 2001.

[70] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

[71] Michael Mitzenmacher and Matteo Dell'Amico. The supermarket model with known and predicted service times. *IEEE Transactions on Parallel and Distributed Systems*, 33:2740–2751, 11 2022.

[72] Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, page 357–368, New York, NY, USA, 2022. Association for Computing Machinery.

[73] Ziv Scully, Isaac Grosof, and Michael Mitzenmacher. Uniform bounds for scheduling with job size estimates. *arXiv preprint arXiv:2110.00633*, 2021.

[74] Maryam Akbari-Moghaddam and Douglas G Down. Seh: Size estimate hedging scheduling of queues. *ACM Transactions on Modeling and Computer Simulation*, 2023.

[75] Tianming Zhao, Wei Li, and Albert Y. Zomaya. Real-time scheduling with predictions. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 331–343, 2022.

[76] Yossi Azar, Eldad Peretz, and Noam Touitou. Distortion-Oblivious Algorithms for Scheduling on Multiple Machines. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 248. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 12 2022.

[77] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor scheduling for real-time systems*. Springer, 2015.

[78] Miguel Alcon, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J Cazorla. Timing of autonomous driving software: Problem analysis and prospects for future solutions. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 267–280. IEEE, 2020.

[79] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No.98EX168)*, pages 172–179, 1998.

[80] Jorge Real and Alfons Crespo. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[81] Luca Santinelli, Giorgio C. Buttazzo, and Enrico Bini. Multi-moded resource reservations. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 37–46. IEEE Computer Society, 2011.

[82] Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. In *Proceedings - Real-Time Systems Symposium*, pages 286–295. IEEE, 1998.

[83] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems*. Springer, 2005.

[84] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[85] Moris Behnam, Insik Shin, Thomas Nolte, and Mikael Nolin. SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems. In Christoph M. Kirsch and Reinhard Wilhelm, editors, *Proceedings of the 7th ACM & IEEE international conference on Embedded software - EMSOFT '07*, pages 279–288, New York, New York, USA, 2007. ACM Press.

[86] Rafia Inam. *Hierarchical scheduling for predictable execution of real-time software components and legacy systems*. PhD thesis, Mälardalen University, December 2014.

[87] Rodrigo Santos, Giuseppe Lipari, Enrico Bini, and Tommaso Cucinotta. On-line schedulability tests for adaptive reservations in fixed priority scheduling. *Real-Time Systems*, 48(5):601–634, 2012.

[88] Wei-Ju Chen, Peng Wu, Pei-Chi Huang, Aloysius K Mok, and Song Han. Online reconfiguration of regularity-based resource partitions in cyber-physical systems. *Real-Time Systems*, pages 1–44, 2021.

[89] Vladimir Nikolov, Stefan Wesner, Eugen Frasch, and Franz J. Hauck. A hierarchical scheduling model for dynamic soft-realtime systems. *Leibniz International Proceedings in Informatics, LIPIcs*, 76(01):71–723, 2017.

[90] Yusen Li, Xueyan Tang, and Wentong Cai. On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 2–11, 2014.

[91] Yusen Li, Xueyan Tang, and Wentong Cai. Dynamic bin packing for on-demand cloud resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):157–170, 2015.

[92] Yossi Azar and Danny Vainstein. Tight bounds for clairvoyant dynamic bin packing. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–21, 2019.

[93] Gordana Dodig-crnkovic. Scientific methods in computer science. In *In Proc. PROMOTE IT 2002, 2nd Conference for the Promotion of Research in IT at New Universities and at University Colleges in*, pages 22–24, 2002.

[94] José María López, José Luis Díaz, and Daniel F. García. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real Time Syst.*, 28(1):39–68, 2004.

[95] Shaik Mohammed Salman, Alessandro V Papadopoulos, Saad Mubeen, and Thomas Nolte. Multi-processor scheduling of elastic applications in compositional real-time systems. *Journal of Systems Architecture*, 122:102358, 2022.

[96] Shaik Mohammed Salman, Alessandro Vittorio Papadopoulos, Saad Mubeen, and Thomas Nolte. Evaluating dispatching and scheduling strategies for firm real-time jobs in edge computing. In *IECON 2023-49th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6. IEEE, 2023.

[97] Shaik Mohammed Salman, Alessandro Vittorio Papadopoulos, Saad Mubeen, and Thomas Nolte. Dispatching deadline constrained jobs in

edge computing systems. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2023.

[98] Shaik Mohammed Salman, Van-Lan Dao, Alessandro Vittorio Papadopoulos, Saad Mubeen, and Thomas Nolte. Scheduling firm real-time applications on the edge with single-bit execution time prediction. In *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 207–213. IEEE, 2023.

[99] Shaik Salman, Thomas Nolte, Alessandro Papadopoulos, and Saad Mubeen. Taming tardiness on parallel machines: Online scheduling with limited job information. Technical report, Mälardalen Real-Time Research Centre, Mälardalen University, October 2024.

II

Included Papers

Chapter 8

Paper A: Multi-Processor Scheduling of Elastic Applications in Compositional Real-Time Systems

Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte.
Journal of Systems Architecture 122 (2022): 102358.

Abstract

Scheduling of applications modeled according to the periodic and sporadic task model under hierarchical and compositional real-time systems has been widely studied to provide temporal isolation on shared resources. However, for some real-time applications which are amenable to a certain amount of variation in their timing behaviour while still maintaining their functional correctness, these tasks models are pessimistic. The elastic tasks model addresses this pessimism by allowing the timing requirements of the application's tasks to be specified as a range of values instead of a single value. While the scheduling of elastic applications on dedicated resources has received considerable attention, there is limited work on scheduling of such applications under hierarchical and compositional settings.

In this paper, we evaluate different earliest deadline first scheduling algorithms to schedule elastic applications in a minimum parallelism supply form reservation on a multiprocessor system. Our evaluation shows that the proposed approach provides similar performance as compared to the current state-of-art algorithms for scheduling elastic applications on dedicated processors but with reduced algorithmic complexity.

8.1 Introduction

The elastic task model enables convenient modelling of real-time applications that can tolerate some amount of variation in their timing behaviour (changes in interarrival times or changes in execution times) while still maintaining their functional correctness. In uniprocessor systems and multi-processor systems where the resources are fully dedicated to an elastic application, the variation can be managed online by modifying the utilization of the application's tasks to ensure schedulability. Scheduling multiple elastic applications on a shared resource, however, requires the use of reservation-based mechanisms to minimize the impact of such variations on the co-scheduled applications. While different solutions have been proposed for scheduling applications with tasks specified according to the periodic or the sporadic task model under reservation-based mechanisms, there is limited work related to scheduling of elastic applications under reservation schemes. In this context, we propose a scheduling framework for executing elastic applications under the minimum-parallelism periodic resource supply model (MPS) on multi-processor systems. While the elastic task model allows defining the task parameters to capture the variable timing requirements of the application, the minimum-parallelism periodic resource supply model [1, 2] provides a relatively simpler mechanism to reserve the resource supply on a multi-processor system. Moreover, under certain period assignment constraints, the MPS model dominates other comparable state-of-art techniques [2].

Concretely, in this paper, we address the following questions:

Q1 Given an application with elastic tasks, what is a feasible bandwidth required to schedule the applications in an MPS reservation on a multiprocessor system?

Q2 Given a fixed bandwidth MPS reservation, what mechanism can the elastic application tasks adopt to remain schedulable?

Q3 Given an elastic application, can a schedulable reservation be found if the application requests for a modified bandwidth reservation?

Q4 What is the trade-off between the proposed approach and the current state-of-art approaches?

8.2 Proposed System Model

We consider the scheduling of a real-time application specified according to the elastic task model with each task having an implicit deadline(See Section. 8.2.1). At the system level, we assume that the CPU resources are made available to each application according to the multi-processor Minimum Parallelism Supply Model (MPS) [2, 1] (see Section 8.2.2). We assume that each application provides its own local scheduling mechanism, based on the partitioned dynamic-priority scheduling implementing the Earliest Deadline First (EDF) policy. The dedicated full processors do not need a system-level scheduler but rather use the application provided scheduler. The partial processor can be managed by any scheduler which can ensure that the partial processor provides supply according to the periodic resource model [3].

8.2.1 The Basic Elastic Task Model

We define an elastic application $\mathbf{A}$ as a set of n tasks where each task is specified as $\tau_i = \{C_i, T_i^{min}, T_i^d, T_i^{max}, E_i\}$. Here, C_i is the Worst-Case Execution Time (WCET) of the task while T_i^{min} and T_i^{max} specify the minimum and the maximum interarrival time between consecutive jobs of τ_i. T_i^d represents the desired period of the task τ_i. The elastic co-efficient E_i determines the flexibility of the task τ_i to change in its period [4]. For instance, if E_i is defined to be in the range $[0, 1]$, $E_i = 0$ ensures that the task's desired period cannot be modified, implying that $T_i^{min} = T_i^d = T_i^{max}$. Similarly, $E_i = 1$ indicates that the task's period can be modified to take up values up to its maximum period. Further, the elastic coefficient acts as a weighting factor, determining the extent to which its utilization can be reduced in relation to other tasks. We use T_i (without any postscript) to indicate the current inter-arrival time of τ_i. An example of an elastic taskset is shown in Table 8.1. Here, the period of the task τ_1 can be extended up to its maximum period, while the task τ_5 can only execute with its desired period.

Task ID	**WCET**	T_i^{min}	T_i^d	T_i^{max}	E_i
τ_1	4	40	120	240	1
τ_2	8	40	80	360	0.75
τ_3	12	240	240	480	0.5
τ_4	12	200	240	600	0.25
τ_5	4	40	40	40	0

Table 8.1: An Elastic Task Set.

The utilization of a task τ_i is defined as $U_i = \frac{C_i}{T_i}$. The minimum and maximum utilization of each task is represented by $U_i^{min} = \frac{C_i}{T_i^{max}}$ and $U_i^{max} = \frac{C_i}{T_i^{min}}$, respectively. The desired utilization is represented by $U_i^d = \frac{C_i}{T_i^d}$. The desired application utilization is given by the sum of the desired utilization of the individual tasks. i.e., $U^d = \sum_{i=1}^{n} U_i^d$. Similarly, the minimum and maximum application utilization is given by $U^{min} = \sum_{i=1}^{n} U_i^{min}$ and $U^{max} = \sum_{i=1}^{n} U_i^{max}$. The relative deadline of each job of an elastic task is equal to its current period at runtime. If a task requests for a change in its current period, its desired utilization U_i^d is updated and a schedulable utilization for rest of the tasks is calculated. An elastic application is said to be schedulable if $U^d \leq U^{ub}$, where U^{ub} is the utilization upper-bound for a given scheduling algorithm. It is assumed that the deadline is elastic-implicit. Note that when the utilization of a task is changed to accommodate increased utilization of another task, the reduced utilization is used to check for schedulability instead of the initial desired period.

8.2.2 Minimum-Parallelism Resource Supply Model

The resource supply provided to the application is based on the minimum parallelism resource supply model [1, 2]. Here the resource supply is provided by reserving m dedicated processors for an application and at most one periodic resource model based partial processor [3] for the exclusive use of the application. Here, the partial processor is defined as $\Gamma(\Theta, \Pi)$, where Θ is the periodic resource allocation time and Π is the resource period. Essentially, the periodic

Task ID	U_i^d
τ_1	0.55
τ_2	0.50
τ_3	0.50
τ_4	0.45
τ_5	0.25
τ_6	0.15

Table 8.2: taskset with desired utilization.

resource Γ provides an application with Θ time units of CPU time every Π time units. The utilization of the partial resource supply is defined as $U_\Gamma = \frac{\Theta}{\Pi}$.

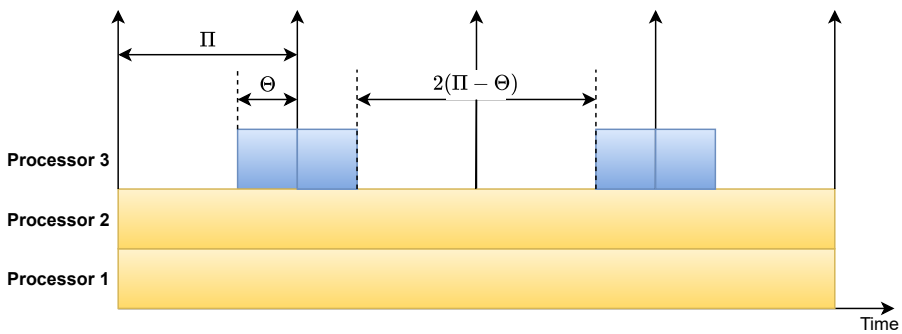


Fig. 8.1: Minimum Parallelism Resource Supply Model.

Fig. 8.1 shows the resource supply provided according to the MPS model with 3 processors. Here, processor 1 and processor 2 are fully dedicated to the application, while processor 3 is available partially and modelled according to the periodic resource model. In the worst case, i.e., when the resource budget is exhausted at the beginning of its period and the next instant it is available close to the end of the next period, the no supply interval of the partial processor is equal to $2(\Pi - \Theta)$ [3].

Generating the resource supply reservation parameters: As the MPS model requires reserving m full processors and one partial processor, we need to identify the number of dedicated processors along with the resource supply parameters of the partial processor necessary to schedule the application. We identify these values based on the initial desired utilization. As we consider partitioned EDF scheduling, we apply the reasonable allocation decreasing (RAD) partitioning heuristic specified in [5]. For example, consider the taskset with its desired utilization in Table 8.2. The tasks τ_1 and τ_4 are assigned to the same processor since their utilization sums up to one. Similarly τ_2 and τ_3 are assigned to another processor. As τ_5 and τ_6 have their utilization much less than the utilization limit for a fully dedicated processor under EDF, they are assigned a partial processor. We apply the method described in [3] to identify the capacity and the period of the partial processor to schedule τ_5 and τ_6.

Schedulability Bounds: For the dedicated processors, we assume the uniprocessor utilization bound of one for partitioned EDF. For the partial processor, we use the utilization bound defined according to the periodic resource model. Particularly, the application tasks assigned to the partial processor with utilization $U_{\mathbf{A}}$ are schedulable under EDF scheduling policy if the partial processor resource supply utilization satisfies Eq. (8.1) (Eq. 21 in [3]).

$$U_{\Gamma,EDF}(k) = \frac{(k+2).U_{\mathbf{A}}}{k+2(U_{\mathbf{A}})}, \tag{8.1}$$

where,

$$k = \max\left\{k \in \mathbb{Z} \middle| (k+1)\Pi - \Theta - \frac{k\Theta}{k+2} < T^{min}\right\} \tag{8.2}$$

and T^{min} is the smallest period among the tasks assigned to the partial processor.

8.3 Proposed Solution

As the minimum parallelism resource supply model allows at most one partial processor, the remaining resources are provided by dedicated processors.

Therefore, we first consider scheduling of elastic applications on dedicated multiprocessors. In this section, we summarize the current state-of-art solution to the scheduling problem based on the partitioned heuristics proposed by Orr et al. [6]. We then extend this solution to include the partial processor.

8.3.1 Scheduling Elastic Applications on Dedicated Multi-Processors

The schedulability of elastic applications on dedicated multi-processors was evaluated by Orr et al. [6] using both global as well as partitioned scheduling approaches. Their work highlighted the relatively better schedulability performance of the partitioned approach when compared to global scheduling for elastic applications. We summarize their approach below and refer to it as the global re-partitioning approach in the rest of the paper.

The Global Re-Partitioning Approach: This approach involves iteratively compressing the utilization and re-partitioning the taskset until a schedulable partition is found. The algorithm iterates over the values in the range $[0, \Phi]$, where Φ is calculated according to (8.4), to find the smallest value of λ such that the utilization assigned to each task according to (8.3) can result in a schedulable partition. During each iteration, it assigns a value to λ and calculates the utilization to be assigned to each task. For this set of utilization values, it then applies a predetermined partitioning heuristic such as the "reasonable allocation decreasing(RAD)"[5], where the tasks are ordered according to monotonically decreasing utilization values and each task is then assigned to the processor on which it fits. The task to processor assignment can be based on

1. First Fit: Assign the task to the first processor on which it fits. i.e., the task meets its schedulability conditions.

2. Best Fit: Assign the task to the processor with minimum remaining capacity after its allocation.

3. Worst Fit: Assign the task to the processor with maximum remaining capacity after its allocation.

Task ID	U_i^{min}	U_i^d	U_i^{max}	E_i
τ_1	0.25	0.55	0.75	1
τ_2	0.25	0.45	0.65	1
τ_3	0.25	0.50	0.75	1
τ_4	0.25	0.50	0.75	1

Table 8.3: An Elastic Task Set

If a task remains unassigned, the value of λ is incremented by a granularity constant ϵ and the process is repeated until either all tasks are assigned to processors, resulting in a successful allocation, or the value of λ exceeds Φ, resulting in failure.

$$U_i = \max(U_i^d - \lambda E_i, U_i^{min}) \tag{8.3}$$

$$\Phi = \max\left(\frac{U_i^d - U_i^{min}}{E_i}\right) \tag{8.4}$$

An Example: Consider an elastic taskset with values given in Table.8.3 that needs to be scheduled on a 2 core processor. According to the first fit heuristic and considering the desired utilization values, the tasks τ_1 and τ_2 are assigned to core 1 while τ_3 and τ_4 are assigned to core 2. At runtime, suppose that the task τ_1 requests for a new utilization equal to 0.65. The algorithm tries to find a suitable value for λ such that a schedulable partition is found. Observing the utilization values in this example, one can notice that the new utilization request of the task τ_1 could have been easily accommodated by changing the utilization of task τ_2 from its desired value to its minimum utilization value without the re-partitioning step. Based on this observation, we propose a new algorithm that applies the utilization modification algorithm on a per-core basis and only re-partitions the taskset if the per-core approach fails to successfully accommodate the requests for increased utilization values.

Utilization Modification on a Single Core: While the re-partitioning approach is relatively straightforward to implement, one disadvantage of this approach is that if a taskset is not schedulable, the global re-partitioning can cause

Task ID	U_i^{min}	U_i^d	U_i^{max}	E_i
τ_1	0.25	0.55	0.75	1
τ_2	0.35	0.45	0.65	1
τ_3	0.15	0.50	0.75	1
τ_4	0.15	0.50	0.75	1

Table 8.4: An Elastic Task Set with Unschedulable Demand

multiple task migrations resulting in additional book-keeping overhead. Further, if the temporal isolation strategies such as cache partitioning are part of an overall solution, especially those based on task-based cache allocation, then task migrations may require re-partitioning of the caches adding to the overhead. One way to minimize such overheads is to limit the utilization modification to tasks running on the same processor or the partial processor as the task requesting the increased utilization.

An Example: While the request for increased utilization of the task τ_1 in the previous example could be successfully managed following the per-core utilization approach, this approach can fail to find a schedulable solution for other requests and tasksets. To illustrate this, consider the elastic taskset as given in Table. 8.4, which is similar to the previous example but with modified minimum utilization values. Based on the desired utilization values, the tasks τ_1 and τ_2 are assigned to core 1 while τ_3 and τ_4 are assigned to core 2. At runtime, suppose that the task τ_1 requests for a new utilization equal to 0.75. Applying the per-core approach would require the utilization of the task τ_2 to be reduced to 0.25. Since the minimum utilization of task τ_2 is greater than the required value, this results in an unschedulable condition. If re-partitioning was applied, however, the request could be successfully handled by assigning τ_1 and τ_3 to the same core and reducing the utilization τ_3 to 0.25. Similarly, τ_2 and τ_4 could be assigned to the same core to ensure schedulability.

Based on these observations, we describe next a combined approach that exploits the advantages offered by the per-core approach while improving the range of schedulable elastic tasksets through limited re-partitioning.

The Combined Approach: Although the original re-partitioning approach finds schedulable solutions for most of the evaluated scenarios(See Section. 8.4), the associated overheads can be minimised by first applying the utilization modification algorithm to the tasks running on the same processor as the task requesting for increased utilization to find a schedulable utilization on that processor. Since the complexity of the utilization modification algorithm is a function of the number of tasks and the number of processors, by reducing the number of tasks whose utilization should be adjusted, the efficiency of the algorithm can be improved. Indeed, if a schedulable utilization is not found, then re-partitioning considering all the cores and the complete taskset cannot be avoided and as such, the overheads related to task migration (and if applicable, cache partitioning) remain the same as in the original approach. The general steps for the combined approach are as follows:

1. Order the taskset either in an (i) arbitrary manner, or (ii) monotonically increasing utilization, or (iii) monotonically decreasing utilization.

2. Order the processors in some arbitrary manner.

3. Allocate tasks to processors according to a "Reasonable Allocation" scheme.

4. When a task makes a request for increased utilization, apply the iterative utilization modification algorithm only to tasks on the same core.

5. If such a request fails, apply the iterative utilization modification algorithm including re-partitioning by considering all the cores.

The combined approach is able to find schedulable solutions similar to the original re-partitioning approach but with improved efficiency since the re-partitioning step is applied only if the per-core utilization modifications fail. In some of the evaluated scenarios, the results indicate per-core utilization modifications are sufficient to keep the application schedulable while completely avoiding the re-partition-ing step. In limited cases where the per-core utilization fails, the re-partitioning approach successfully finds a schedulable partition.

8.3.2 Extension to Minimum Parallelism Resource Supply Model

The approaches discussed in the previous section considering dedicated processors can be extended to the minimum parallelism resource supply reservation model with some minor modifications. One of the key distinguishing characteristics between the MPS reservation model and the fully dedicated processors is the presence of the partial processor. Since the partial processor is only available periodically, under worst-case conditions, there can be a no supply interval equal to $2(\Pi - \Theta)$ (See Fig. 8.1). Any task with a period less than this no supply interval cannot be allocated to the partial processor. As a consequence, the re-partitioning step should not only consider the utilization but also the period of the tasks. Moreover, in the case where both per-core utilization modification, as well as re-partitioning, fail due to the period constraints, depending on the availability of resources on the core executing the partial processor, a re-allocation of the bandwidth on the partial processor can be considered. The general steps of the proposed approach are as follows:

1. Order the taskset (i) arbitrary manner (ii) monotonically increasing desired utilization or (iii) monotonically decreasing desired utilization.

2. Order the processors in some arbitrary manner.

3. Allocate tasks to processors according to a "Reasonable Allocation" scheme.

4. if any task remains unallocated, generate resource supply parameters for the partial processor according to the initial desired utilization values of the unallocated tasks.

5. When a task makes a request for increased utilization, apply the iterative utilization modification algorithm only to tasks on the same core (or partial processor as applicable).

6. if such a request fails, apply the iterative utilization modification algorithm by including re-partitioning considering all the dedicated cores. Include partial processor only if the failing request belongs to the partial processor.

7. During re-partitioning, only allocate tasks whose periods are greater than the worst-case no-supply duration on the partial processor while ensuring that the total utilization of the task assigned to the partial processor remains below the partial processor utilization according to Eq.(8.1).

8. If this request fails, modify the bandwidth of the partial processor if resources are available and repeat steps 3 and 4.

8.4 Evaluation

We evaluated the proposed approach for the case of fully dedicated processors by considering randomly generated tasksets with a varying number of tasks and utilization values along with a different number of processors and partitioning heuristics. Here we present the results of the 16 such configurations as detailed in Table 8.5. For each configuration, we measured the number of successful requests along with the computation time (based on the Query Performance Counter provided by the Windows OS) for a prototype implementation of the proposed algorithms.

Taskset Generation: The desired utilization of each task of a taskset was generated using the algorithm proposed by Griffin et al. [7]. The minimum utilization was generated by subtracting a randomly generated percentage (taken from a uniform distribution) from the desired utilization. Similarly, the maximum utilization was generated by adding a randomly generated percentage to the desired utilization while ensuring that the maximum utilization for each task was under one. Each request for increased utilization was generated by selecting a random task and a uniformly distributed random utilization value from within the task's desired and maximum utilization. The elastic co-efficient was assigned to each task by randomly generating real values taken from a uniform distribution between [1-10].

Configurations: To compare the performance of the proposed approaches with a state-of-art method, we evaluated different configurations by varying the number of processors, and tasks along with different percentage limits on the

Configuration	Maxmin	No.of tasks	No.of Processors
1	1, 0.5	100	2
2	1, 0.5	100	4
3	1, 0.5	100	8
4	1, 0.5	100	16
5	1, 0.5	200	2
6	1, 0.5	200	4
7	1, 0.5	200	8
8	1, 0.5	200	16
9	0.5, 0.5	100	2
10	0.5, 0.5	100	4
11	0.5, 0.5	100	8
12	0.5, 0.5	100	16
13	0.5, 0.5	200	2
14	0.5, 0.5	200	4
15	0.5, 0.5	200	8
16	0.5, 0.5	200	16

Table 8.5: Different Configuration Settings.

utilization values to define the maximum and minimum utilization of each task. For the desired utilization, the maximum utilization of each task was set to be not greater than 0.5. Further, the tasks were ordered according to monotonically decreasing utilization values. The first fit approach was used as the task to processor assignment strategy. Table 8.5 shows the different configurations where the column "maxmin" indicates the maximum and minimum values used to define the utilization values. For example, the values $(1, 0.5)$ indicate that the maximum utilization that a task can have is in the interval $[U_d, U_d + (1 \cdot U_d)]$, while the minimum utilization a task can have is in the interval $[U_d - (0.5 \cdot U_d), U_d]$. The values under the column "configuration" refer to the respective (row) configurations settings and are used as identifiers on the x-axis in the associated graphs.

8.4.1 Results

We evaluated the different approaches for schedulability along with the computation times under different configuration requirements. For each configuration, we generated ten different tasksets and for each taskset, we simulated 1000 requests for utilization modifications. For each utilization request, we assigned the new utilization values chosen from a uniform distribution between the desired utilization and the maximum utilization. Moreover, each task requesting increased utilization was chosen randomly.

Schedulability: The schedulability of the different configurations for the three methods is shown in Fig. 8.2. Here, GP refers to the global re- partitioning approach of ORR et al., while PCM refers to the per-core utilization modification approach and CA refers to the combined approach of per-core modifications and re-partitioning. The results show that the combined approach has a 100 percent success value while the per-core modifications have a success value of 95 percent in the worst-case indicating that in most cases, the per-core modification is sufficient. In case of failure, the limited re-partitioning associated with the combined approach can find a schedulable solution. The variation in successful requests for certain configurations is mostly due to the randomness in the generated tasksets and the tasks requesting for the modified utilization.

For configurations with a reduced number of tasks (Table 8.6), the schedulability performance of the per-core approach was around 93 percent in the worst-case scenario, which was relatively worse than compared to tasksets with a larger number of tasks while the other two approaches had 100 percent success.

Computation time: For each configuration, we measured the computation time required to find a schedulable solution and to report a failure if no solution is found through a prototype implementation of the proposed approaches on a Windows system. Fig. 8.3 shows the average computation time per request for each of the different configurations. The average was calculated by measuring 10000 requests. The results indicate that the per-core and the combined approach perform relatively better compared to the global re-partitioning

Configuration	Maxmin	No.of tasks	No.of Processors
1	1, 0.5	20	2
2	1, 0.5	20	4
3	1, 0.5	20	8
4	0.5, 0.5	20	2
5	0.5, 0.5	20	4
6	0.5, 0.5	20	8

Table 8.6: Configuration with a smaller number of tasks.

approach. In certain scenarios, the average computation time of the combined approach is 6 times better than the global re-partitioning approach. This improvement is mostly due to the fact that most requests are managed within the core and that re-partitioning is done only if necessary, unlike the global re-partitioning approach where every new request results in re-partitioning.

Fig. 8.4 shows the worst-case computation times taken by the different approaches. The results indicate that for most configurations, the proposed approaches perform better than the global partitioning approach. In certain cases, however, the worst-case performance of the combined approach is similar (or worse in limited scenarios) to the global re-partitioning approach.

For configurations with a reduced number of tasks (Table 8.6), the computation performance of all the approaches was better compared to the configurations with a larger number of tasksets. Comparing the performance of different approaches for the same number of tasks (See Fig. 8.6), the average value of the combined approach was up to two times better than the global partitioning approach. The worst-case performance of the combined approach (See Fig. 8.7) was worse than the global approach for most of the evaluated configurations.

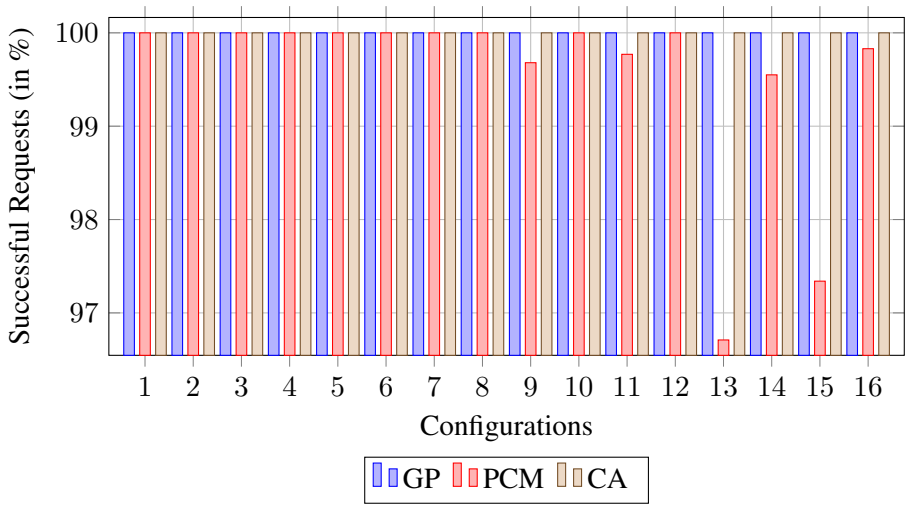


Fig. 8.2: Schedulability Performance of the Different Approaches.

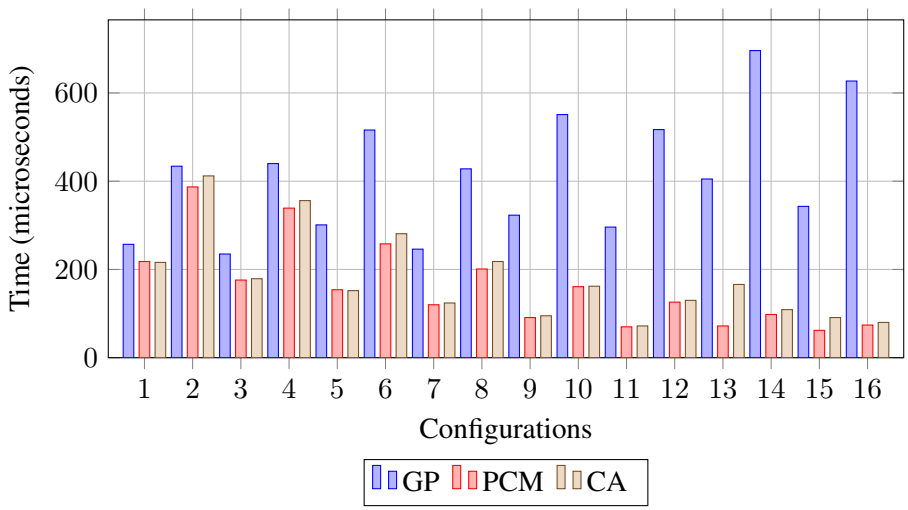


Fig. 8.3: Average Computation Times of the Different Approaches.

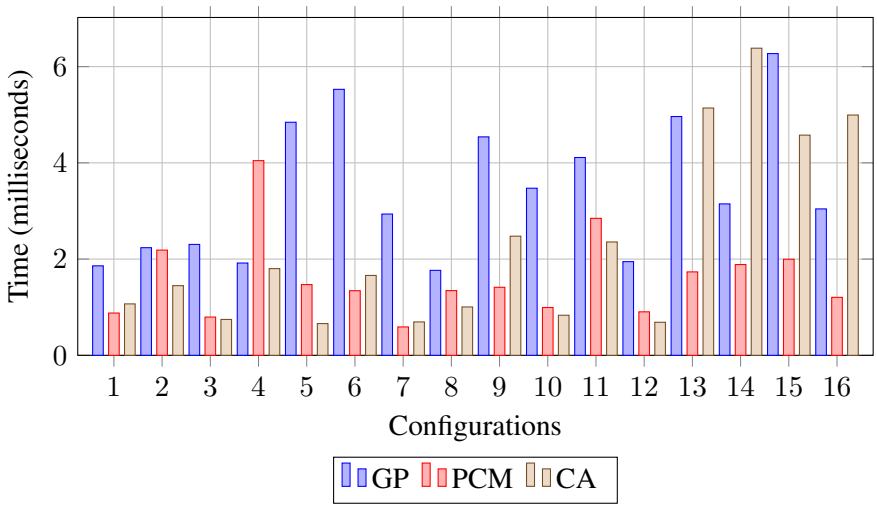


Fig. 8.4: Worst Case Computation Times of the Different Approaches.

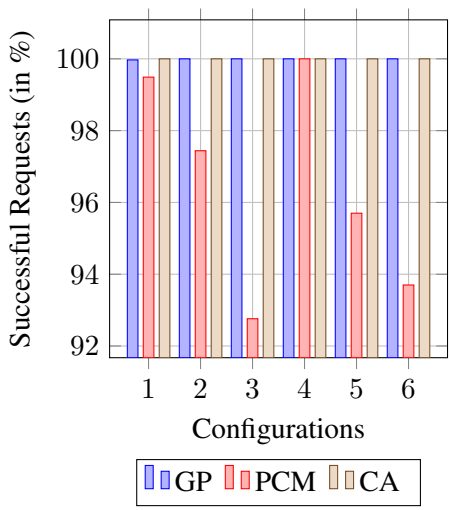


Fig. 8.5: Schedulability Performance of Smaller Tasksets.

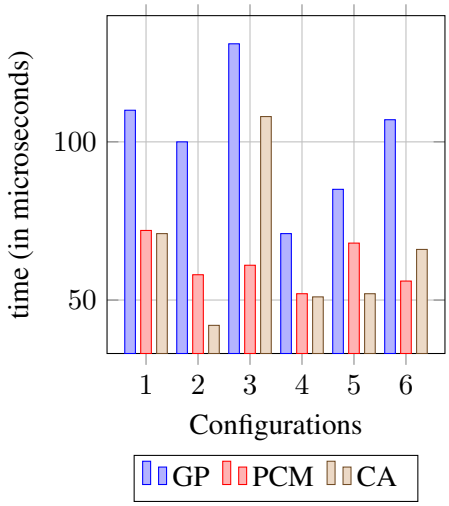


Fig. 8.6: Average Computation Times for Smaller Tasksets.

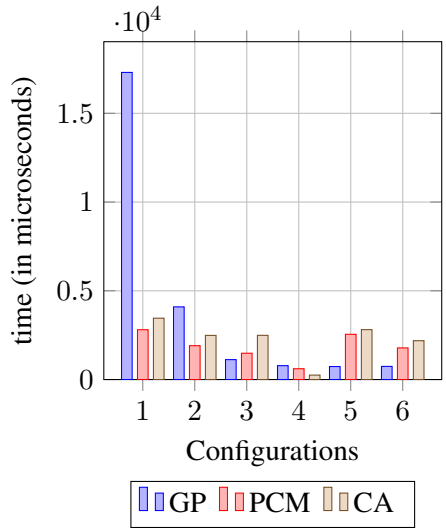


Fig. 8.7: Worst Case Computation Times for Smaller Tasksets.

8.5 Related Work

Hierarchical scheduling of periodic and sporadic real-time applications was encapsulated in a compositional real-time scheduling framework by Lee et al. [3]. In this framework, the computational demand of an application was abstracted with a single demand interface as a pair of capacity and period and the resource supply server was abstracted as a periodic resource model where each server was guaranteed a reserved capacity Θ every Π time units. Easwaran et al. [8] extended the periodic resource model to include the deadline parameter, where each server was guaranteed a reserved capacity Θ within D time units, in every time interval Π. Dewan et al. [9, 10] provided algorithms to find an approximate allocation of bandwidth for a set of periodic and sporadic tasks under the periodic resource model. The periodic resource model was further extended for the case of multiprocessors by Shin et al. [11] where the periodic resource supply model was augmented with a parameter indicating the maximum concurrency with which the resource supply is provided. The tasks assigned to this resource supply were then scheduled using a global scheduler. Bini et al. [12, 13] provide a more expressive and general model for specifying multiprocessor resource supply in the form of the parallel supply function. Lee et al.[14] build upon the multiprocessor resource supply model and provide a cache-aware compositional analysis for the minimum parallelism resource supply form for the global EDF scheduling policy.

The concept of elastic tasks was introduced by Buttazzo et al. [15] to model applications whose computational demands can occasionally exceed the available capacity by allowing the application to modify the demand by changing the frequency of its jobs through an elastic coefficient. This was extended to address resource sharing within the elastic task model in [4]. Guangming [16] provided an earlier time for accelerating and adding tasks for the elastic scheduling approach. Chantem et al. [17, 18] reformulated the problem as a quadratic optimization problem and showed that the original elastic tasks compression algorithm was indeed a solution to solving a quadratic problem. Tian et al. [19] extended the modified problem to include a "Quality-of-Control" metric as a part of the objective function of the quadratic optimization problem. More recently, Orr et al. [20, 6] provided algorithms to schedule sequential

elastic tasks on multiprocessor systems and further extended the concept of the elastic task to federated DAG-based parallel task systems in [21, 22]. Beccari et al. [23, 24] provided alternative algorithms to schedule similar applications by expressing the task period ranges in a linear programming formulation.

Another commonly used approach to schedule application tasks with timing variability is to modify the resource reservations. Thiele et al. [25] provide an online reconfiguration algorithm for the constant bandwidth server. Khalilzad et al. [26] proposed an adaptive hierarchical scheduling model to accommodate the adaptive behaviour of the periodic and sporadic tasks by changing the bandwidth allocation. In contrast, this paper assumes that a bandwidth allocated for a server under the periodic resource model remains constant and that the workload within the server can be adapted according to the elastic task model. However, if the elastic assignment fails, a request for a new bandwidth allocation will be made. We note that the proposed solution does not take into account possible bandwidth reclamation or mixed-criticality-based approaches to assign new bandwidths if no schedulable allocation can be made. Instead, we leave it to the individual application to handle such failures. For uniprocessor systems, we provide a method to schedule elastic applications within a periodic resource supply model reservation in [27].

8.6 Conclusion

Scheduling of elastic applications on reservation-based multiprocessors systems has not been extensively studied. To address this, we proposed a scheduling framework based on the minimum-parallelism resource reservation model and compared different partition-based EDF scheduling mechanisms. The proposed methods introduce a relatively intuitive approach to scheduling elastic tasks under reservation schemes on multiprocessors combining the advantages of the simplicity offered by the MPS reservation and flexibility of the elastic tasks. The evaluation results indicate the proposed methods outperform the current state-of-art approaches on average, while in the worst case, none of the approaches outperforms the other. In future work, we intend to investigate methods to improve the worst-case performance and evaluate the results on a real system.

8.7 Acknowledgement

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation, and by the Swedish Knowledge Foundation (KKS) under the projects FIESTA, HERO and DPAC.

Bibliography

[1] Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 191–200, 2008.

[2] Kecheng Yang and James H. Anderson. On the dominance of minimum-parallelism multiprocessor supply. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 215–226. IEEE, 2016.

[3] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):30:1–30:39, May 2008.

[4] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[5] José María López, José Luis Díaz, and Daniel F. García. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real Time Syst.*, 28(1):39–68, 2004.

[6] James Orr and Sanjoy Baruah. Algorithms for implementing elastic tasks on multiprocessor platforms: a comparative evaluation. *Real-Time Systems*, 57:227–264, 2021.

[7] David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 76–88, 2020.

[8] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 129–138. IEEE Computer Society, 2007.

[9] Nathan Fisher and Farhana Dewan. A bandwidth allocation scheme for compositional real-time systems with periodic resources. *Real Time Syst.*, 48(3):223–263, May 2012.

[10] Farhana Dewan and Nathan Fisher. Bandwidth allocation for fixed-priority-scheduled compositional real-time systems. *ACM Trans. Embed. Comput. Syst.*, 13(4):91:1–91:29, March 2014.

[11] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 181–190, 2008.

[12] Enrico Bini, Marko Bertogna, and Sanjoy Baruah. Virtual multiprocessor platforms: Specification and use. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 437–446, 2009.

[13] Artem Burmyakov, Enrico Bini, and Eduardo Tovar. Compositional multiprocessor scheduling: the GMPR interface. *Real-Time Syst.*, 50(3):342–376, 2014.

[14] Meng Xu, Linh T.X. Phan, Insup Lee, Oleg Sokolsky, Sisu Xi, Chenyang Lu, and Christopher Gill. Cache-aware compositional analysis of real-time multicore virtualization platforms. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 1–10, 2013.

[15] Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 286–295. IEEE, 1998.

[16] Qian Guangming. An earlier time for inserting and/or accelerating tasks. *Real-Time Syst.*, 41(3):181–194, 2009.

[17] Thidapat Chantem, Xiaobo Sharon Hu, and M. D. Lemmon. Generalized elastic scheduling. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 236–245, 2006.

[18] Thidapat Chantem, Xiaobo Sharon Hu, and Michael D. Lemmon. Generalized elastic scheduling for real-time tasks. *IEEE Transactions on Computers*, 58(4):480–495, 2009.

[19] Yu Chu Tian and Li Gui. QoC elastic scheduling for real-time control systems. *Real-Time Syst.*, 47(6):534–561, 2011.

[20] James Orr, Chris Gill, Kunal Agrawal, Jing Li, and Sanjoy Baruah. Elastic scheduling for parallel real-time systems. *Leibniz Transactions on Embedded Systems*, 6(1):5:1–5:14, 2019.

[21] James Orr, Christopher D. Gill, Kunal Agrawal, Sanjoy K. Baruah, Christian Cianfarani, Phyllis Ang, and Christopher Wong. Elasticity of workloads and periods of parallel real-time tasks. In *International Conference on Real-Time Networks and Systems (RTNS)*, pages 61–71. ACM, 2018.

[22] James Orr, Johnny Condori Uribe, Christopher D. Gill, Sanjoy K. Baruah, Kunal Agrawal, Shirley Dyke, Arun Prakash, Iain Bate, Christopher Wong, and Sabina Adhikari. Elastic scheduling of parallel real-time tasks with discrete utilizations. In *International Conference on Real Time Networks and Systems (RTNS)*, pages 117–127. ACM, 2020.

[23] Giuseppe Beccari, Stefano Caselli, Monica Reggiani, and Francesco Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 21–28. IEEE Computer Society, 1999.

[24] Giuseppe Beccari, Stefano Caselli, and Francesco Zanichelli. A technique for adaptive scheduling of soft real-time tasks. *Real Time Syst.*, 30(3):187–215, 2005.

[25] Pratyush Kumar, Nikolay Stoimenov, and Lothar Thiele. An algorithm for online reconfiguration of resource reservations for hard real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 245–254. IEEE, 2012.

[26] Nima Moghaddami Khalilzad, Thomas Nolte, Moris Behnam, and Mikael Åsberg. Towards adaptive hierarchical scheduling of real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 1–8, 2011.

[27] Mohammed Salman Shaik, Saad Mubeen, Filip Marković, Alessandro V. Papadopoulos, and Thomas Nolte. Scheduling elastic applications in compositionalreal-time systems. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021.

Chapter 9

Paper B: Evaluating Dispatching and Scheduling Strategies for Firm Real-Time Jobs in Edge Computing

Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte.
In IECON 2023-49th Annual Conference of the IEEE Industrial Electronics Society (IECON 2023)

Abstract

We consider the problem of on-arrival dispatching and scheduling jobs with stochastic execution times, inter-arrival times, and deadlines in multi-server fog and edge computing platforms. In terms of mean response times, it has been shown that size-based scheduling policies, when combined with dispatching policies such as join-shortest-queue, provide better performance over policies such as first-in-first-out. Since job sizes may not always be known apriori, prediction-based policies have been shown to perform reasonably well. However, little is known about the performance of prediction-based policies for jobs with firm deadlines. In this paper, we address this issue by considering the number of jobs that complete within their deadlines as a performance metric and investigate, using simulations, the performance of a prediction-based shortest-job-first scheduling policy for the considered metric and compare it against scheduling policies that prioritize based on deadlines (EDF) and arrival times (FIFO). The evaluation indicates that in under-loaded conditions, the prediction-based policy is outperformed by both FIFO and EDF policies. However, in overloaded scenarios, the prediction-based policy offers slightly better performance.

9.1 Introduction

Edge computing architectures allow real-time system designers to deploy algorithms on high-performance edge computing servers to improve response times and reduce energy consumption in embedded devices[1, 2, 3]. In addition, these edge deployments may need to support several different devices necessitating a multi-server system design. For such multi-server systems, dispatching and scheduling algorithms are important in satisfying response time requirements[4]. Specifically, when the objective is mean response times and the processing times of individual jobs are known apriori, multi-server variants of shortest-remaining-processing-time (SRPT) and preemptive shortest job first (SJF) are optimal under a central queue assumption [5]. Similarly, when jobs are dispatched on arrival, Mitzenmacher et al.[6] showed that SRPT and SJF outperformed the FIFO policy when combined with the join-shortest-queue (JSQ) dispatching policy. A limitation of these policies is that they require the knowledge of processing times of each job before its completion, which may not always be possible[7]. As an alternative, usage of predicted values has been investigated to achieve improved performance in terms of mean response times compared to size-oblivious policies such as first-in-first-out (FIFO) ordering[8, 6]. In a large-scale queuing system, Mitzenmacher [8] studied the impact of single-bit predictors that can indicate whether a job's processing time is above or below some threshold. They found that such predictors can provide benefits similar to those achievable with knowledge of exact processing times for Poisson arrivals and certain processing time distributions. While mean response time can be a useful measure for certain applications, some real-time applications impose constraints in terms of deadlines, where the output is only valid if it is generated before its deadlines. However, they can tolerate some missed deadlines [9]. Additionally, some of these algorithms may exhibit stochastic behavior in terms of processing times as well as inter-arrival times(For example, See [10, 11]) The performance evaluation of scheduling policies based on predicted processing times for such workloads with firm deadline requirements has received limited attention.

We address this by first considering the presence of a dual-bit job size predictor that can classify an incoming job into one of the four job size classes:

small, medium, large, and very large. We choose a dual-bit prediction approach based on the intuition that the accuracy of such predictors may be better than the accuracy of predictors estimating individual job processing times. Secondly, we use the information the dual-bit predictor provides with a preemptive shortest-job-class-first (PSJF) scheduling policy. This policy orders jobs according to their job size classes, and jobs in each class are in FIFO order. Furthermore, we consider an on-arrival dispatching policy and leave evaluation of central queue approaches for future work.

Concretely, we formulate the following questions and address them using simulations.

1. What is the impact of on-arrival job acceptance and rejection based on response-time estimation of pending jobs on achievable throughput under various load conditions?

2. How does a dual-bit prediction-based PSJF scheduling policy compare against EDF, FIFO, and SRPT scheduling policies in terms of achievable throughput under various load conditions?

3. What is the impact of on-arrival server selection policy on achievable throughput under various load conditions when using dual-bit prediction-based PSJF scheduling policy?

Our evaluation indicates that prediction-based PSJF provides no significant advantage over FIFO and EDF scheduling policy for the considered settings and under-loaded scenarios. However, under fully loaded and sustained overload conditions, it performs better than FIFO and EDF when used with an estimation-based admission policy.

9.2 Related Work

In on-arrival dispatching systems, dispatching policies determine the server on which an incoming job will be executed. Dispatching policies such as JSQ and its variants that require the knowledge of the number of pending jobs in each server is optimal with respect to mean response times [4]. However, gathering

this information may introduce overheads depending on the number of servers and the network traffic. Alternatively, policies such as round-robin (RR) that do not require knowledge of pending jobs dispatch incoming jobs in a cyclic order. Consequently, they do not have the overhead associated with policies that require information about the pending jobs on each server. Several other policies, such as join-the-idle-queue and join-below-threshold, have been proposed to balance the trade-off between overheads and response times [12, 13].

Many studies have explored the potential for enhancing algorithm performance through machine-learned advice or predictions, including classical algorithms for online scheduling and load-balancing [14]. These prediction-augmented algorithms have been evaluated through competitive analysis under both accurate and possibly incorrect predictions [8, 6, 15, 13]. In online scheduling, some researchers have considered predicting job execution times [13, 16] and the ordering of jobs [15]. Mitzenmacher et al. [6] demonstrated via simulations that the benefits of using predictions in large distributed systems were retained if the predictions were reasonably precise. Based on the evaluations, they proposed selecting servers with the least number of pending jobs and using the predicted shorted processing job first policy for use in actual systems. Zhao et al. [13] extended the RMLF algorithm to use predicted job execution times. The prediction enhanced algorithm achieved performance close to that of SRPT when the prediction error was small and better than RMLF when the error was large. However, designing highly accurate predictors that predict the exact size of a job may be difficult. Consequently, we consider dual-bit predictors that coarsely classify an incoming job into one of the four distinct job classes.

In single-server settings, Gao et al. [17] developed scheduling strategies for firm semi-periodic real-time tasks. They introduced three control parameters to decide at run-time whether to interrupt a job before its deadline and considered four admission policies. Our work differs in that we consider a multi-server setting and estimate response times using the job execution time distribution and the number of pending jobs on a specific server on job arrival and allowing admitted jobs to stay in the queue until completion or reaching their deadline.

Several works within queuing theory analyzed the performance of EDF under different scenarios. Abhaya et al. solve a set of linear equations to

calculate the mean delay for M/G/1/./EDF [18]. Kargahi et al. [19] provide bounds for estimating the deadline miss probabilities for M/M/k/./NEDF+G and M/M/1./EDF+G assuming a single queue system, unlike the dispatch on arrival policy we considered. Kargahi provided an analytical method in [20] to show the performance of parallel EDF queues for JSQ, the minimal expected value of unfinished work(MED), and threshold-based dispatching strategy but without any arrival time acceptance or rejection.

We previously evaluated the performance of a dispatching policy that relies on single-bit predictions of job processing times in conjunction with a non-preemptive FIFO scheduling policy where all jobs had a fixed relative deadline. [21]. Similarly, we evaluated the percentage of missed deadlines when jobs have individual deadlines using preemptive EDF policy in [22]. The work in the paper differs from our previous work in that we consider a prediction-based policy and compare it with EDF and FIFO ordering.

9.3 System Model

9.3.1 Job Model

Jobs arrive online following a Poisson process with an arrival rate λ adjusted accordingly to generate desired system load. We assume a Poisson process since this has been extensively used in the literature in analyzing queuing systems and models quite well the use case we consider. i.e., requests for job executions can arrive from several users, and each such user may have different inter-arrival times. The processing time of each job is drawn from a trimmed and discretized exponential distribution with a mean of 10 and lower bounded with value 1 and upper bounded by value 100. (i.e., ten times the mean value). While this may not be a realistic representation of many real-world workloads, it has been widely used in the literature in queuing systems, and the loss of accuracy compared to true exponential distribution may be acceptable since we only consider the average throughput rather than numerically precise response times. Additionally, each job's relative deadline d_i is drawn from a uniform distribution D with a range between five to ten times the mean value of the execution time distribution. We choose the considered ranges due to negative

results associated with low laxity systems[15]. The relative deadline is revealed when the job arrives. Each job is assigned an absolute deadline on its arrival.

9.3.2 Server Model

We consider a network of homogeneous servers. Each server has its own queue and executes jobs assigned to it according to the considered scheduling policy. When a job arrives, a dispatching policy selects a server and accepts or rejects the job based on the pending workload on the selected server. Each accepted job is added to the queue of the selected server. The jobs in the queue are ordered based on the scheduling policy. If a new job has a shorter deadline than the currently executing job, the scheduler preempts that job, adds it to its own queue, and starts executing the new job.

Server Selection Policy: We evaluate JSQ and RR server selection policies. Under JSQ, whenever a new job arrives, the server with the least number of pending jobs is selected, while in RR, the server is selected cyclically. We combine these policies with an online schedulability test to enable admission time control. If a job is deemed to be schedulable, it is immediately sent to the selected server and is rejected otherwise.

Admission Policy: A schedulability test decides whether a job can be successfully scheduled on a server, given a scheduling policy and information about the pending jobs on the server. As jobs in our system are bound by a deadline, we must determine whether the job can meet its deadline on the selected server. Utilization-based schedulability tests that rely on worst-case processing time values can be used if the service cannot tolerate any deadline miss. However, if over-provisioning is a problem and deadline misses are tolerated, a low-overhead but less accurate test may be useful. If the jobs satisfy such a schedulability test, they are accepted into the system.

We now describe the policies used to admit or reject the jobs. We consider three policies based on how the job processing times are considered, (i) mean-approximation policy and (ii) clairvoyant policy, and (iii) admit-all policy.

Mean-approximation policy: In this policy, we use mean μ of the processing time distribution to estimate the response time f_i of a newly arrived job on the selected server i. If the number of pending jobs on this server is given by N_i, the estimated response time is given by

$$f_i = (N_i + 1) \cdot \mu. \tag{9.1}$$

Clairvoyant policy: In this policy, we assume the knowledge of exact processing times. The response time f_i on any server i is given by

$$f_i = x_j + \sum_{k=0}^{N_i} x_k, \tag{9.2}$$

where x_k is the exact processing time of each job k assigned to server i and x_j is the processing time of the newly arrived job.

For both the estimation methods, the admission test returns true if the following condition is satisfied:

$$f_i \leq d_i. \tag{9.3}$$

Admit-all policy: This policy dispatches each incoming job to a selected server and does not reject any job.

Scheduling Policies

We consider four different policies depending on the workload parameters, (i) FIFO policy, prioritizing jobs based on their arrival times, (ii) EDF, prioritizing jobs based on their deadlines (iii) PSJF, prioritizing jobs based on their predicted processing times and (iv) SRPT, prioritizing jobs based on their true remaining processing time.

- FIFO policy prioritizes jobs according to their arrival times, with ties broken arbitrarily. It executes jobs in a non-preemptive manner.
- EDF policy prioritizes jobs according to their absolute deadlines with preemptions. i.e., a newly added job can preempt a running job if its absolute deadline is lower than that of the running job.

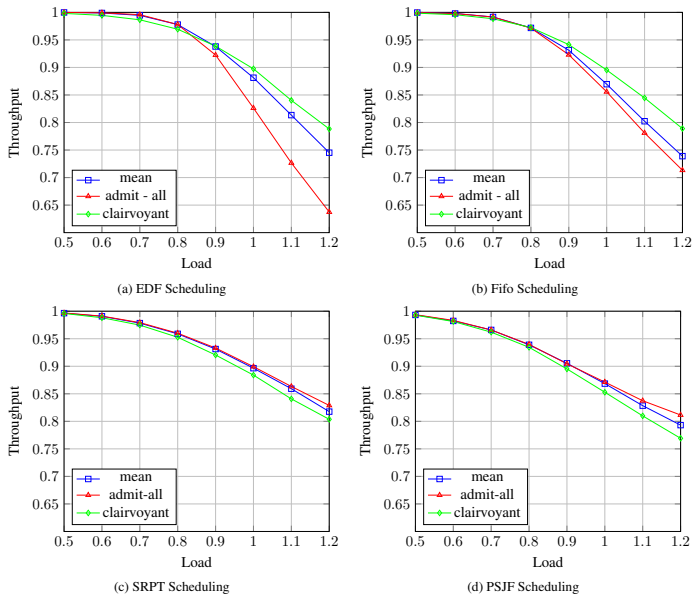


Fig. 9.1: Comparison of throughput under three different admission policies for round-robin dispatching and various scheduling policies

- SRPT policy prioritizes jobs according to their remaining processing times with preemptions. Although this policy requires the exact processing time to be known, we use this policy as a baseline to compare against the size-aware prediction-based PSJF policy.

- PSJF policy prioritizes jobs according to their predicted job classes, with jobs from the shortest class executed first with preemptions enabled. Jobs within each class are in FIFO order.

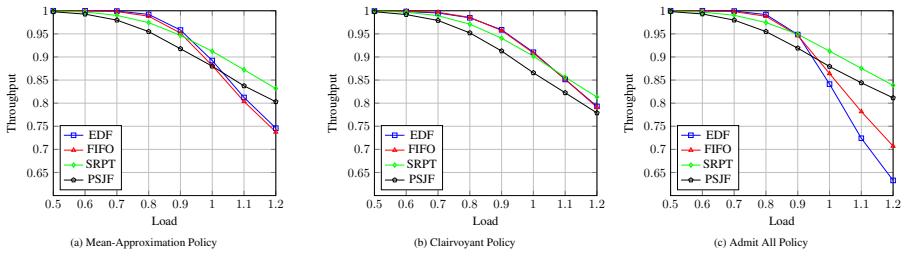


Fig. 9.2: Comparison of throughput under different scheduling and admission policies for RR dispatching

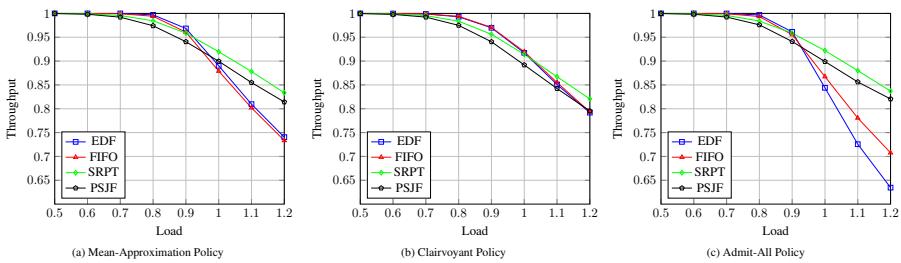


Fig. 9.3: Comparison of throughput under different scheduling and admission policies for JSQ dispatching

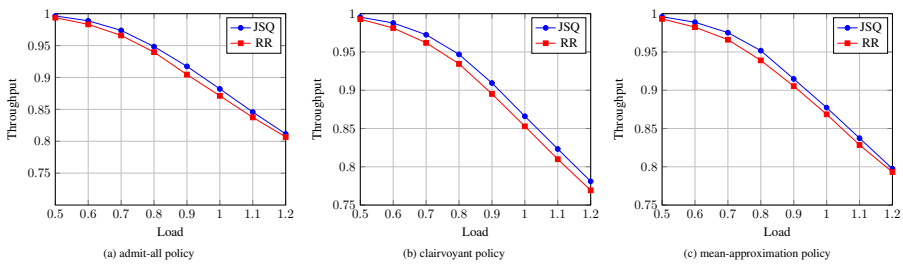


Fig. 9.4: Comparison of throughput under different dispatching schemes for PSJF scheduling policy

9.4 Simulation Methodology

We compare the performance of the scheduling policies for (trimmed) exponentially distributed job processing times, Poisson inter-arrival times, and uniformly distributed deadlines. We simulate 10000 time units and measure throughput as the ratio of the number of jobs completed within their deadlines and the total number of jobs released during this simulation interval. We repeat the simulation over ten runs and present the average throughput calculated over these ten runs.

Load generation: We generate jobs based on the load following Eq. (9.4), where ρ is the load, μ is the mean processing time, and λ is Poisson arrival rate per server, and n is the number of servers. Specifically for the results presented in this paper, we fix μ to 10, set n to 2 and 4, and calculate λ for fixed values of load ρ. Specifically, we set ρ between 0.5 and 1.2 and incremented in steps of 0.1.

$$\rho = \frac{\mu}{n \cdot \lambda} \tag{9.4}$$

Assigning processing times: For each job, we first assign the true processing time generated according to the trimmed exponential distribution with μ set to ten. The lower bound is set to one, and the upper bound is set to one hundred. As we use integer time units for stepping through the simulation, we use the $ceil$ function to convert the generated floating point value to an integer. We note that this discretization changes the mean of the distribution and deviates from the true exponential distribution. However, considering a large sample size, i.e., a hundred thousand samples, this deviation can be ignored without significantly impacting the observed results.

Assigning predicted processing times: As we assume a dual-bit predictor, which classifies jobs into four distinct classes, defining the thresholds that can guide the classification is necessary. As a first approach, we consider quartile values of the exponential distribution and use them as threshold values. Once the true processing time has been assigned, we assign the predicted processing time equal to the discrete version of the quartile values. Specifically, we use

values 3, 7, 17, and 47, the rounded quartile values of the trimmed exponential distribution with a mean of 10.

Assigning deadlines: To assign the relative deadline, we first generate a random number from a uniform distribution of a range of 5 to 10. This generated random number is multiplied by the mean of the processing time distribution, and the result is assigned as the deadline. If the true processing time value is greater than the assigned deadline, we reset the processing time to the mean of the distribution while retaining the deadline. We do this since this allows us to keep the generated jobs, as it eliminates the problem of having unschedulable jobs even before they are released.

9.5 Evaluation

9.5.1 Performance of Admission Policies

We compared the performance of the scheduling policies with and without admission control policies. As a baseline, we considered a clairvoyant policy that knows the exact size of each pending job on a chosen server. The estimated response time of an incoming job is then calculated using eq.(9.2). We also considered a low complexity mean approximation policy where the response time is estimated using eq.(9.1). In addition to this, we considered the scenario where all incoming jobs are admitted and dispatched to specific servers depending on the dispatching policy. The achieved throughput with RR dispatching when using two servers is shown in Fig. 9.1. When using EDF as the scheduling policy, rejecting jobs using the admission control policies has a limited impact on achievable throughput when the load is below 0.9. However, the benefits of admission control are seen when the load is increased to 1.0, with even the mean-approximation policy achieving a six percent higher average throughput than the admit-all policy. The difference, however, is reduced under the FIFO scheduling policy. Similarly, when considering SRPT and the prediction-based PSJF policy, the impact of the admission control policies remains negligible until a load value of 0.9. When the load increases to 1.0 and above, admitting all jobs and mean-approximation policy-based admission

control provide slightly better throughput than the clairvoyant policy. Based on these observations, we can conclude that on-arrival admission control policies provide no significant advantage when the system load is below 0.9 when combined with any of the considered scheduling policies. If the system load is above 0.9, admission control policies perform better when combined with size-oblivious EDF and FIFO scheduling policies. When combined with size-aware PSJF and SRPT, it is better to admit all jobs rather than reject some jobs to achieve a slightly better average throughput.

9.5.2 Impact of Scheduling Policy

Fig. 9.2 and Fig. 9.3 provide a comparison of the achieved throughput of the considered scheduling policies when combined with different admission control policies for RR and JSQ dispatching, respectively. When using the mean approximation policy and admit-all policy, we can observe that as load increases, the throughput reduces for all the scheduling algorithms. When the load is below 0.9, size-oblivious scheduling policies perform slightly better than the baseline SRPT policy, while the prediction-based PSJF performs the worst. However, when the load is close to 1.0, all the policies perform similarly. Under overload conditions, PSJF performs better than both EDF and FIFO, while SRPT performs the best. The performance of EDF is worst under the admit-all policy. When using the clairvoyant policy, PSJF performs worst at high loads, with EDF and FIFO performing better than SRPT.

9.5.3 Performance of Dispatching Policies

We compared the impact of JSQ and RR dispatching policies on the performance of various scheduling and admission control policies. Fig. 9.4 shows the throughput achieved with PSJF scheduling and various admission control policies. We observe that the difference in performance due to the dispatching policy is almost negligible, with JSQ only slightly outperforming RR. This is seen consistently across all load values. This behavior is also consistent for size-aware and size-oblivious scheduling policies, irrespective of the admission policy.

9.5.4 Discussion

We compared the performance of various combinations of admission control and scheduling policies, including a coarse prediction-based shortest job policy for applications whose jobs exhibit variability in inter-arrival times, processing times, and deadlines. The results show that for non-asymptotic conditions, i.e., when the number of servers is limited to two and four, PSJF and SRPT provide no advantage over EDF and FIFO policies when the load is below 0.9. However, size-aware policies provide better throughput under overloaded conditions, with even the coarse-grained PSJF performing better than both FIFO and EDF when combined with mean approximation and the admit-all admission policies. Additionally, JSQ and RR perform similarly for all combinations with no significant difference in performance. This indicates that the overheads of JSQ can be avoided with very little loss by choosing RR. Moreover, suppose it can be established that the load in the system will stay below 0.9. In that case, the simple FIFO policy with an admit-all policy can be used instead of the other policies with relatively higher computational complexities. Another interesting observation is that using size-aware policies for admission control and scheduling provides worse performance. However, it must be noted that we only considered the exponential distribution for processing time distribution, and the observations may not apply to a different distribution. In addition to this, we assumed an ideal predictor that is fully accurate with zero inaccurate job size classifications. This may not be realizable in practice, and further investigation is needed to study the impact of inaccuracies. Another missing parameter is network communication and the lack of consideration of end-to-end deadlines, which may be important in practical systems.

9.6 Conclusion

We considered the problem of on-arrival dispatching and scheduling firm real-time jobs in multi-server settings. We evaluated the performance of the predicted job scheduling policy using coarse-grained predictions in terms of average throughput using simulations under non-asymptotic conditions. Our evaluation shows that the prediction-based size-aware policy does not offer sig-

nificant benefits compared to policies that prioritize based on deadlines or arrival times in under-loaded conditions. However, in overloaded scenarios, it performs slightly better than other policies. In summary, our study provides valuable insights into selecting dispatching and scheduling policies for edge computing systems to meet the needs of firm real-time applications.

Bibliography

[1] Josip Zilic, Atakan Aral, and Ivona Brandic. Efpo: Energy efficient and failure predictive edge offloading. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 165–175, 2019.

[2] Josip Zilic, Vincenzo De Maio, Atakan Aral, and Ivona Brandic. Edge offloading for microservice architectures. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '22, page 1–6, New York, NY, USA, 2022. Association for Computing Machinery.

[3] Matthijs Jansen, Auday Al-Dulaimy, Alessandro V. Papadopoulos, Animesh Trivedi, and Alexandru Iosup. The spec-rg reference architecture for the edge continuum, 2022.

[4] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[5] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. Srpt for multiserver systems. *ACM SIGMETRICS Performance Evaluation Review*, 46(2):9–11, 2019.

[6] Michael Mitzenmacher and Matteo Dell'Amico. The supermarket model with known and predicted service times. *IEEE Transactions on Parallel and Distributed Systems*, 33:2740–2751, 11 2022.

[7] Mor Harchol-Balter and Ziv Scully. The most common queueing theory questions asked by computer systems practitioners. *ACM SIGMETRICS Performance Evaluation Review*, 49(4):3–7, 2022.

[8] Michael Mitzenmacher. Queues with small advice. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, pages 1–12. SIAM, 2021.

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

[10] Miguel Alcon, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. Timing of autonomous driving software: Problem analysis and prospects for future solutions. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 267–280, 2020.

[11] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 286–295. IEEE Comput. Soc, 2-4 Dec. 1998.

[12] Xingyu Zhou, Fei Wu, Jian Tan, Yin Sun, and Ness Shroff. Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–30, 2017.

[13] Yecheng Zhao, Runzhi Zhou, and Haibo Zeng. Design optimization for real-time systems with sustainable schedulability analysis. *Real-Time Systems*, 58(3):275–312, 2022.

[14] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

[15] Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In *Proceedings of the 34th ACM Symposium*

on Parallelism in Algorithms and Architectures, SPAA '22, page 357–368, New York, NY, USA, 2022. Association for Computing Machinery.

[16] Ziv Scully, Isaac Grosof, and Michael Mitzenmacher. Uniform bounds for scheduling with job size estimates. *arXiv preprint arXiv:2110.00633*, 2021.

[17] Yiqin Gao, Guillaume Pallez, Yves Robert, and Frederic Vivien. Dynamic scheduling strategies for firm semi-periodic real-time tasks. *IEEE Transactions on Computers*, 72:55–68, 1 2023.

[18] Vidura Gamini Abhaya, Zahir Tari, Panlop Zeephongsekul, and Albert Y. Zomaya. Performance analysis of edf scheduling in a multi-priority preemptive M/G/1 queue. *IEEE Transactions on Parallel and Distributed Systems*, 25(8):2149–2158, 2014.

[19] Mehdi Kargahi and Ali Movaghar. A method for performance analysis of earliest-deadline-first scheduling policy. *Journal of Supercomputing*, 37(2):197–222, 2006.

[20] Mehdi Kargahi and Ali Movaghar. Dynamic routing of real-time jobs among parallel edf queues: A performance study. *Computers & Electrical Engineering*, 36(5):835–849, 2010.

[21] Shaik Mohammed Salman, V Lan Dao, S Mubeen, AV Papadopoulos, and Thomas Nolte. Scheduling firm real-time applications on the edge with single-bit execution time prediction. In *26th International Symposium On Real-Time Distributed Computing (ISORC)*, 2023.

[22] Shaik Mohammed Salman, S Mubeen, AV Papadopoulos, and Thomas Nolte. Dispatching deadline constrained jobs in edge computing systems. In *27th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2023.

Chapter 10

Paper C: Dispatching Deadline Constrained Jobs in Edge Computing Systems

Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte.
In IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA 2023)

Abstract

The edge computing paradigm extends the architectural space of real-time systems by bringing the capabilities of the cloud to the edge. Unlike cloud-native systems designed for mean response times, real-time industrial embedded systems are designed to control a single physical system, such as a manipulator arm or a mobile robot, that requires temporal predictability. We consider the problem of dispatching and scheduling of jobs with deadlines that can be offloaded to the edge and propose DAL, a deadline-aware load balancing and scheduling framework that leverages the availability of on-demand computing resources along with an on-arrival dispatching scheme to manage temporal requirements of such offloaded applications. The evaluation indicates that DAL can achieve reasonably good performance even when execution times, arrival times, and deadlines vary.

10.1 Introduction

Complex real-time systems such as mobile robots and industrial robots traditionally follow an embedded deployment model with software functions running on a shared hardware platform to control a single physical system such as the robot [1]. Several studies have investigated the benefits of a cloud-oriented design that depends on the elastic availability of resources for real-time applications [2, 3, 4, 5]. One approach to realize a cloud-oriented design is to package much of the application software, including the operating system, as a Virtual Machine (VM) or container and deploy it to the cloud or edge infrastructure [6, 7] where each VM or container is responsible for a single physical system. An alternative model is a microservice design, where individual functions have their own resources, such as CPU and memory, and handle requests from multiple physical systems rather than a single system. Several approaches have been proposed for scheduling microservices requests in such a deployment model, which take into account latency requirements but have limited consideration for issues such as satisfying deadline constraints [8, 9, 10]. Within queuing theory, several works have addressed deadline-based scheduling policies such as Earliest-Deadline-First (EDF) for systems with a single queue [11, 12, 13] and for the scenario where the number of available processors is fixed [14, 15, 16].

Additionally, several algorithms that can be offloaded exhibit variability in execution times [17], inter-arrival times [18] as well as in terms of their relative deadlines [19]. For example, while most real-time task models assume that the relative deadline of a real-time task is fixed, for some tasks this deadline may also vary per job [20] as well as during the execution of a job [19]. The inter-arrival time between successive jobs can also vary significantly [18, 21]. For systems such as mobile robots, the inter-arrival time can be a function of the current velocity and the distance they are expected to cover before new data is needed [22].

To address this, we consider an edge architecture that relies on the concept of on-demand processing commonly used in cloud-computing designs [23] and uses existing low-complexity dispatching and scheduling policies. Specifically, we present DAL, a deadline-aware load balancing and scheduling framework

that integrates an on-arrival dispatcher along with an EDF scheduling policy by considering the availability of a processor pool with reserved processors and on-demand processors to reduce the number of jobs that miss their deadlines even with variable execution times, arrival times, and deadlines. To keep the design simple, each DAL instance manages a single microservice and assumes that the computing infrastructure can be viewed as a processor pool with a set of reserved processors and a set of on-demand processors. In the context of controlling real-time physical systems, if we consider the physical system as a client that makes a request for the service offered by the microservice task, a response must be sent to this client within a certain time duration. DAL uses an on-arrival dispatcher that assigns incoming requests to the processor that most likely meets their timing requirements. The dispatcher estimates the time a request will take on a processor before dispatching it. Based on recent work related to redundant designs [24], and overload management in data centers [25], for requests that are deemed to miss their timing requirements, DAL includes a feedback mechanism that notifies the requester of a possible violation of its timing requirement so that it can take remedial action locally. Concretely, we show via simulations that using on-demand processors and a low-complexity admission policy can provide significantly improved performance compared to using only reserved processors for jobs with deadline constraints and variable execution and inter-arrival times.

10.2 Motivation

To further motivate DAL's design choices, we consider path planning [26] and trajectory generation [27] as examples of a microservice and highlight the challenges imposed by such services.

Variable Arrival Times: In a multi-user scenario, requests may come from different clients. Even for periodic requests coming from the same clients, each client may have different periods. For example, if the service computes a trajectory for a robot, a robot moving at a higher velocity can send requests at a higher frequency. To address this issue, we designed DAL to manage variable arrivals with an incoming request dispatcher that can decide whether

the incoming request can be processed within its deadline given the pending requests and the constraints on available resources.

Variable Execution Times: Similar to varying arrival times, the execution time for each request can also vary, even for a service that provides the same functionality for all requests. For instance, a path planning service needs to compute paths where execution times vary depending on the required accuracy, segment lengths, and number of collision checks. For example, Alcon et al. [17] analyzed the variability in execution times of prediction and planning modules of an autonomous driving stack, and found that the variations ranged from a minimum of 25 milliseconds to a maximum of 350 milliseconds, and between 175 and 250 milliseconds respectively. To manage such variability, we design DAL to take advantage of the on-demand availability of resources.

Variable Deadlines: For a service like trajectory planning, each request can have its own deadline. For example, a request may have a shorter deadline if the current velocity of the client robot is higher for the same distance when compared to another request where the client robot is moving slowly. Gog et al. [20] highlighted this in the context of an autonomous driving system while Shih et al. [19] considered such state-dependent deadlines. DAL addresses deadline variability by considering EDF as its scheduling policy as it sorts jobs according to their deadlines.

Latency Violation Feedback: Many real-time systems can tolerate missing deadlines [28, 29, 30] and services like the path planner are no different. If a request misses its deadline, a local planner running on the robot can take over and take corrective action, such as running a local instance of the planner [24], or it can reduce its speed and send a new request with a relaxed deadline. For this reason, DAL is designed to notify a requester if its request cannot be satisfied as estimated by the dispatcher, and subsequently, if it misses the deadline while waiting in the queue. Additionally, DAL deletes the requests that have missed their deadlines to service pending requests and possibly new requests from the same client robot, similar to the analysis in [15].

10.3 Related Work

Kargahi [31] provided an analytical method to show the performance of parallel EDF queues for join-shortest-queue(JSQ) dispatching and a threshold-based dispatching strategy but without any arrival time acceptance or rejection. Wang et al. [23] considered the problem of dispatching and scheduling requests with heterogeneous reserved and on-demand processors where individual requests have maximum waiting time described by an exponential distribution. They provided a mathematical model as well as the multi-queue request scheduling framework that assigns incoming jobs to different queues depending on the type of the processors, followed by the allocation of jobs in queues to specific processors. They also provide a mechanism to decide the number of on-demand processors to be provisioned. Here the on-demand processors are utilized when the queue is full or when the waiting time exceeds the maximum waiting time.Similarly, Meng et al.[32] considered the problem of dispatching and scheduling jobs with arbitrary deadlines and bounded worst-case execution time on a set of reserved processors along with network transmission delays. However, these algorithms may be unsuitable for applications with low latency requirements due to their complexity.

Gao et al. [33] proposed strategies for scheduling firm semi-periodic real-time jobs in single-processor environments. The jobs are released periodically and share the same relative deadline, but their execution times can have arbitrary probability distributions. The researchers explored several optimization criteria, including the Deadline Miss Ratio (DMR). To determine whether a job should be interrupted before its deadline, they introduced three new control parameters at runtime. These parameters include an upper bound on completion times, which is used to drop a job if it cannot be completed by a certain time between periodic inter-arrival time and relative deadline; an upper bound on job execution times, which is used to reject jobs with execution times exceeding a certain value; and an upper bound on waiting time, which is used to drop a job that has waited until a certain bound. They also considered four admission policies, which involve admitting all jobs, admitting jobs until a fixed number of jobs are in the queue, admitting jobs with a fixed probability, and admitting jobs following a repeating pattern. The evaluation of their work revealed that

the most critical control parameter for achieving the best DMR is the upper bound on the waiting time of each job. In contrast to this work, our research utilizes admission policies that estimate the response times based on the job execution time distribution and the number of pending jobs on a particular server. Admitted jobs are allowed to remain in the queue until they are completed or until their deadline.

10.4 System Model

Task Model: We consider a microservice as a task, and each task releases a job of that task when a request arrives. A task is specified by its execution time distribution[1] E, a poisson arrival process with rate λ, and a deadline distribution D. Each job i of the task takes an unknown amount of time E_i from the distribution E and is expected to be completed before a relative deadline d_i drawn from a uniform distribution. The relative deadline d_i is revealed when the request arrives at the dispatcher (Fig.10.1). We assume that the time to decode the deadline information is zero[2]. Each request is assigned an absolute deadline D_i^a by the request decoder at the time it arrives according to eq.(10.1).

$$D_i^a = t_c + d_i, \tag{10.1}$$

where t_c is the time at which the request is decoded.

Processor Pool: We assume that an arbitrary but fixed number of processors are reserved to execute requests of the microservice task. Each job of the task can be executed on any reserved processor. Each of the reserved processors has its own queue with pending jobs ordered according to the EDF scheduling policy. In addition to reserved processors, we also assume that a microservice is deployed on a set of on-demand processors. An on-demand processor can be released if it has no pending jobs of the considered task in its queue. Furthermore, we assume that there is no setup cost associated with on-demand proces-

[1] Same as service time distribution in queuing theory.

[2] In practice, a request arriving at the NIC is processed in FIFO order and may take a non-negligible amount of time before its relative deadline is known.

sors, i.e., when the dispatcher sends a job to an idle on-demand processor, it immediately starts executing the dispatched job.

Dispatch-on-Arrival and Scheduling Policy: Once a request is processed, it is immediately dispatched to the queue of a reserved processor. The processor selection strategy is described in Section 10.5. The jobs in a processor queue are ordered by their absolute deadlines which are calculated using Eq. (10.1). If a new job has a shorter deadline than the currently executing job, the scheduler preempts that job, adds it to its own queue, and starts executing the new job.

10.5 DAL

Scheduling jobs with stochastic parameters with static resource reservations may not provide useful performance unless the reservations are made based on worst-case behavior. For instance, when we evaluated join the shortest queue dispatch policy for exponential arrival and service times with only reserved processors and no on-demand processors, 20 percent of requests missed their deadlines even when another 20 percent of requests were discarded by the dispatcher (see Table 10.5 and Table 10.1). We designed DAL to achieve the goal of successfully completing jobs before their deadlines for microservices with stochastic arrival times, execution times, and deadlines by considering the availability of on-demand resources consistent with the computing model of fog and cloud architectures. DAL's architecture is shown in Fig. 10.1. In the following sections, we describe the various components and policies that DAL employs to achieve this goal.

10.5.1 Processor Pool

DAL's design relies on the concept of a processor pool, which is based on the idea of on-demand availability of processors as supported by cloud and fog computing paradigms but can also work at the edge layer where the number of available processors may be limited. DAL assumes that it has access to fixed number of homogeneous processors at any given time in its processor pool including reserved and on-demand processors. Among these processors,

reserved processors are available for DAL's exclusive use, while on-demand processors may or may not be always available. An on-demand processor is considered available if it is idle when a request is received by DAL's dispatcher, or if it has pending jobs belonging to DAL's jobs and is unavailable when it is executing jobs of a different microservice of lower priority.

Processor Allocation: When a request arrives at DAL's dispatcher, DAL first attempts to send the request to one of the reserved processors. If the dispatcher decides that the request cannot meet its deadline on one of the reserved processors, it searches for an available on-demand processor within the group of on-demand processors. If it finds an available processor, the job is immediately sent to that processor.

Processor Deallocation: Once a request executing on an on-demand processor completes, DAL is expected to release the processor for use by other services. However, instead of releasing the on-demand processor after the request completes, DAL holds the on-demand processor for a configurable duration by speculating on the arrival of another request within the configured duration. If no job is assigned to the processor within this period, it is released back to the processor pool.

10.5.2 Dispatcher

Several dispatch-on-arrival load balancing strategies aim to minimize expected response times for different types of job distributions and assume a fixed number of processors. Common dispatching strategies include join random queue, join shortest queue, and the power-of-d strategy, where d processors are randomly selected and the job is distributed to the processor with the fewest jobs [34, 35]. Since it is known that the performance of the power-of-d policy [34] improves the average response times under different scheduling policies such as first-in-first-out and shortest remaining processing time, DAL combines this policy with a configurable schedulability test to dispatch the jobs. DAL's dispatcher first looks for the reserved processor with the least number of pending jobs and checks whether the incoming job is schedulable. If the job

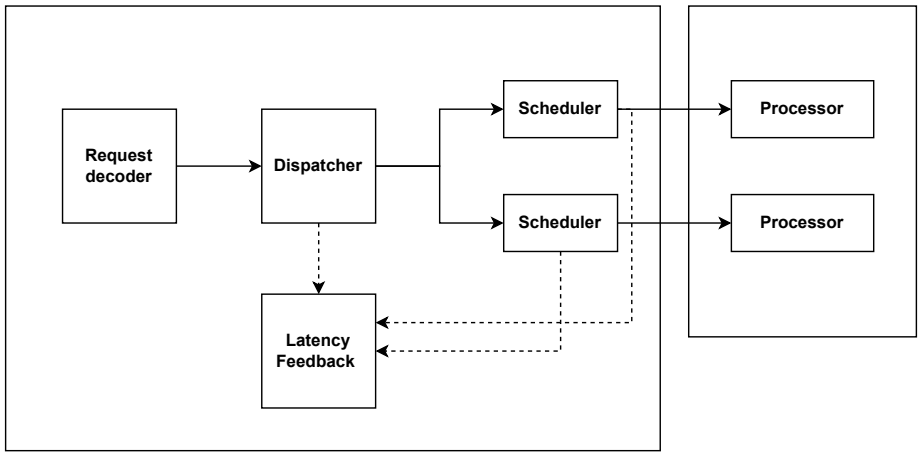


Fig. 10.1: System architecture of DAL.

is deemed to be schedulable, it is immediately sent to that processor. If the job fails the schedulability test, the dispatcher searches for an available processor from the set of on-demand processors, and if it finds one, it assigns the request to that processor. If no such processor is found, it signals the latency feedback component to send a response to the request sender about a potential deadline miss and adds the request to the queue of the processor with the least number of pending jobs among the reserved processors.

***:** Online Schedulability Test A schedulability test decides whether a job can be successfully scheduled on a processor given a scheduling policy and information about the pending jobs on the processor. As jobs in our system are bound by a deadline, we must determine whether the job can meet its deadline on the selected processor. Utilization-based schedulability tests that rely on worst-case execution time values can be used if the service cannot tolerate any deadline miss. Such a test requires us to keep track of all deadlines of pending requests in the queue. However, if over-provisioning is a problem and deadline misses are tolerated, a low-overhead but less accurate test may be useful. DAL dispatches jobs to a processor if the jobs satisfy such a schedulability

test. DAL's schedulability test estimates the response time of an incoming job. If the estimated response time is shorter than its deadline, the incoming job is assumed to pass the schedulability test. Estimating the response time requires information about job execution times and the number of pending jobs in the queue. when using EDF policy, the accuracy of the response time estimation depends on how many pending jobs have a lower absolute deadline on the processor and the probability that future jobs will be assigned to that particular processor and that those future jobs will have a lower absolute deadline than the current job. While information about the deadlines of pending requests can be obtained, the information about the number of future jobs that jump ahead of the incoming job is difficult to predict. Moreover, even if such knowledge is available, as the exact execution time is assumed to be unknown, but only its distribution is known, the estimation depends on which value is chosen as representative of this distribution. To be useful under different distributions, DAL's configurable schedulability test introduces a parameter α that decides the proportion of pending jobs that it considers to have deadlines lower than that of the incoming job. Baldwin et al. show in [36] how such a value can be determined. The influence of α becomes relevant as the size of the pending jobs in the queue increases. To account for the dependence on the distribution of execution time, another parameter β decides which execution time value is used to estimate the response time. This can be derived by applying the central limit theorem [35]. The estimated response time R of the incoming job is given by

$$R = \beta * k * E_m, \tag{10.2}$$

where k is the number of pending requests with a shorter deadline than the incoming job, and E_m is the expected value of the execution time distribution. This is similar to Theorem 4 in [37]. k can be determined either by tracking the deadlines of the pending jobs or from

$$k = \alpha * N, \tag{10.3}$$

where N is the number of pending jobs in the queue. If R is less than its relative deadline d_i, the dispatcher assigns an absolute deadline value to the job according to Eq. (10.1) and adds it to the processor's queue.

10.5.3 Scheduler

DAL instantiates the preemptive EDF scheduling policy on all processors in its processor pool as DAL is designed to manage requests with deadlines. Whenever a reserved processor receives a request from the dispatcher, it starts executing the job if it has no pending requests. If it is currently executing a job, it checks whether the new job has a lower absolute deadline than the job being executed. If it does, the executing job is preempted and the new job is scheduled. otherwise, it sorts the pending requests including the newly arrived job according to their absolute deadlines. Additionally, DAL's per-processor scheduler keeps track of the waiting times of jobs queued in its queue. Whenever a new job arrives or a job is completed, it checks if any of the pending jobs have waiting times that exceed their deadlines. If such jobs exist, it notifies DAL's latency feedback component and deletes the jobs from its queue.

10.5.4 Latency Feedback

As DAL is designed for request-response communication, the requester expects a response from the server. Instead of holding requests that could not be dispatched until a processor is available, DAL notifies the requestor if the request could not be dispatched. It also notifies the requestor when jobs assigned to processors do not complete their execution within the deadline. In the first case, the advantage of early latency violation notification is that it allows the requester to take remedial actions such as locally computing the result on possibly slower computers, while still managing to get the result before the deadline, which would not be possible if it received the notification after the deadline. In addition, such early notification may also allow the requester to modify its requested deadline and send a new request. For example, if the requestor is a system such as a mobile robot, it can reduce its velocity and send an updated request with a new relaxed deadline. The significance of such a feedback mechanism becomes even more apparent when DAL cannot access any of the on-demand processors.

10.6 Evaluation

We conducted simulations in various scenarios to evaluate the performance of DAL. The assessment criteria were missed deadlines, dropped jobs, slowdown ratio, and the number of processors utilized during the simulation period. A job is deemed to have missed its deadline if the sum of its waiting time and executed time exceeds the deadline, including jobs deleted before receiving any processing time. We consider a job to be dropped if the dispatcher anticipates that it cannot meet its deadline on any processor in its processor pool upon arrival. We define slowdown as the ratio of a job's actual execution time to its response time, considering only jobs that meet their deadlines. The percentage of missed deadlines and dropped jobs was evaluated as the ratio of the number of jobs that missed their deadlines or were dropped over the total number of jobs released during the simulation time. For the evaluation, we assume that all on-demand processors are always available. This assumption enables us to compare the best possible result achievable against the scenario where only reserved processors are utilized.

10.6.1 Simulation Methodology

To generate job execution times, we utilized the exponential distribution class template of the C++ library. The generated float values were rounded off to the nearest largest integer using the CEIL function of the C++ standard library. The release times of the jobs were generated following a Poisson process. For most experiments, we adjusted the arrival rate to 90 percent of the service rate, i.e., the inverse of the expected value of the execution time distribution, for different numbers of reserved processors. We set the mean of the execution time distribution to 40 and set α to 1, resulting in all pending requests being treated as those with shorter deadlines. We also let β equal 1, setting the estimated execution time of each job equal to the mean of its execution time distribution. We assigned deadlines to each job by multiplying its actual execution time with a value from a uniform distribution in the range $[2, 10]$. We note that this approach may not be practical since the actual execution time is typically unknown. However, it avoids jobs with deadlines shorter than their execution times. While we varied the number of reserved processors between one and

Table 10.1: Performance of different on-arrival dispatch policies without on-demand processors.

Dispatch Policy	**Missed Deadlines %**	**Dropped Jobs%**
JSQ, load =90 percent	19.6209	19.6047
FF, load =90 percent	63.2401	0.0133305
JSQ, load = 50 percent	3.52308	3.50832
FF , load = 50 percent	49.13	0

eight, we present the results only for the case where the number of reserved processors was set to four, unless otherwise specified, as the results had similar trends for different numbers of processors. The simulation proceeded stepwise with a tick value of 1 for a duration of 1 million ticks and the execution and arrival times were set as multiples of the tick value. We assumed that the overhead of DAL's implementation was zero, although this assumption was ideal, allowing us to evaluate the approaches under consideration without considering implementation-specific details.

10.6.2 Performance with Reserved Processors

We evaluated the performance of DAL's dispatching and scheduling policy by only considering the reserved processors with exponentially distributed execution times and four reserved processors. Using the dispatcher's shortest queue approach, we found that approximately 20 percent of the total released jobs missed their deadline after being admitted, while an additional 20 percent of the jobs were dropped on-arrival by the admission control policy when the system was 90 percent loaded.

As an alternative dispatching solution, we considered a First-Fit (FF) dispatching approach that uses an arbitrary but static processor ordering and assigns jobs to the first processor on which an incoming job is deemed schedulable. With the FF policy, we observed that only 0.01 percent of the jobs were deemed unschedulable by the admission control policy, but about 63 percent of the jobs missed their deadlines, as shown in Table 10.1. Although the JSQ dispatch policy is better than the FF approach, just over 60 percent of the released jobs managed to complete before their deadlines.

Table 10.2: Performance of DAL under low load conditions.

Dispatch Policy	Missed Deadlines%	Dropped Jobs%
JSQ with Release	0.181869	0
JSQ without Release	0.00184169	0

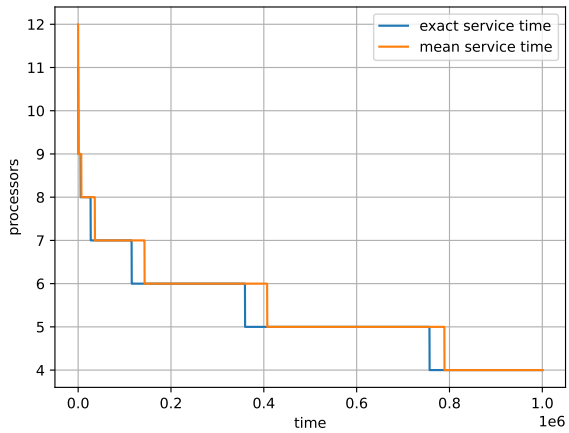
For lower loads, i.e., 50 percent load, about 92 percent of the requests managed to meet their deadlines with JSQ policy, while the FF policy achieved a success rate of less than 50 percent. These observations indicate that using only reserved processors may not be sufficient for deploying microservices with real-time requirements at higher load values.

10.6.3 Performance with On-Demand Processors

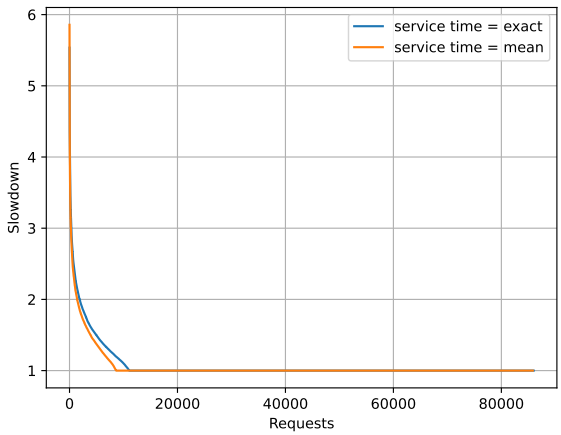
We evaluated the performance of DAL's dispatching and scheduling policy with on-demand processors for exponentially distributed execution times with four reserved processors.

Low Load Scenario: With the system load set to 50 percent, we observe that up to 99.92 percent of the jobs are able to meet their deadlines when idle processors are released if they do not have any pending jobs. This is further improved to 99.99 percent when on-demand processors are not released. Table 10.2 shows the percentage of deadlines missed when on-demand processors are released if they do not have any pending requests and when on-demand processors are not released back to the processor pool. However, this improvement comes at a cost of increased resource usage as more than four on-demand processors are retained for 90 percent of the simulation duration, as seen in Fig. 10.4. If idle processors are released, on-demand processors are used for only 20 percent of the simulation time.

Based on these observations, we can conclude that releasing on-demand processors as soon as they become idle can provide reasonable performance in terms of slowdown as well as in successfully completing up to 99.92 percent of the jobs before deadlines, with lower resource usage.

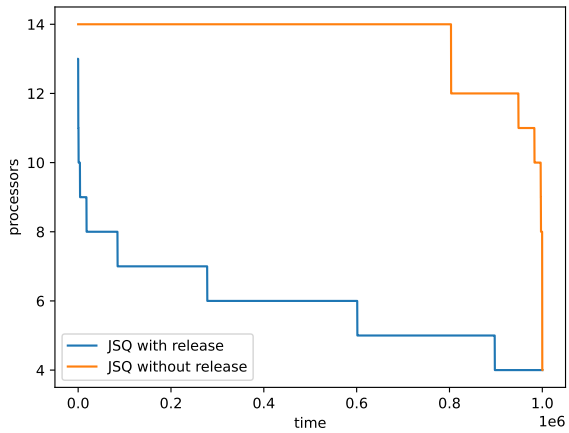


(a) Processor Usage

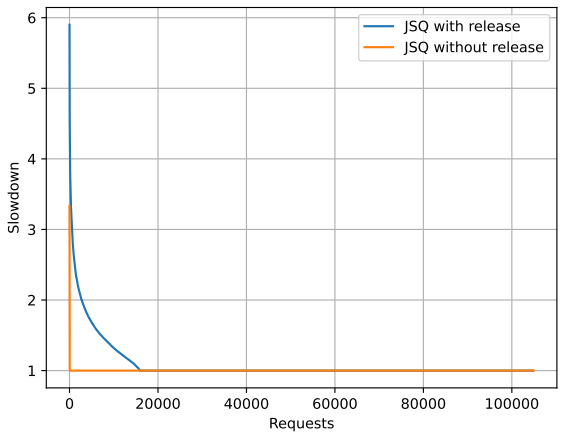


(b) Slowdown

Fig. 10.2: Comparison of processor usage and slowdown ratio when using approximated execution time values versus exact execution time values in a high load scenario.

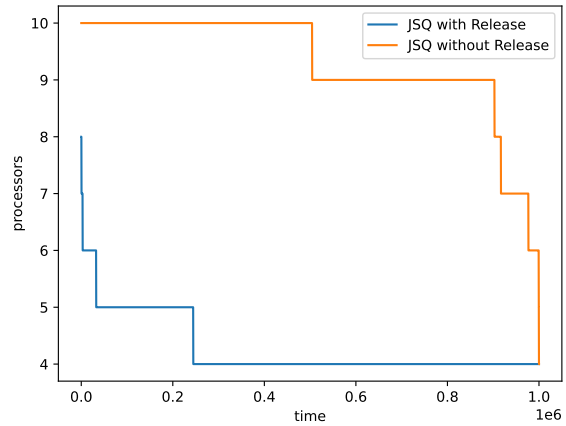


(a) Processor Usage

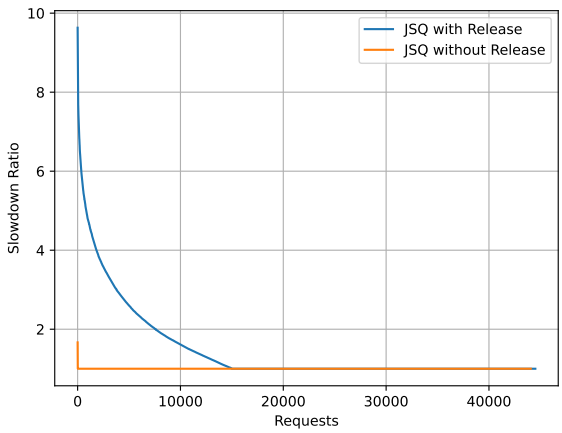


(b) Slowdown

Fig. 10.3: Comparison of processor usage and slowdown ratio when on-demand processors are not released and when released as soon as they are idle in a high load scenario.

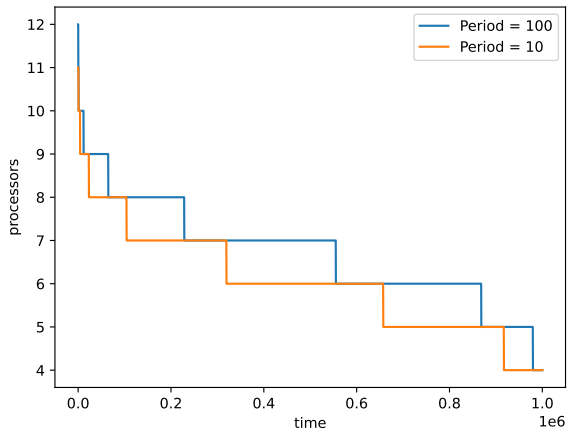


(a) Processor Usage

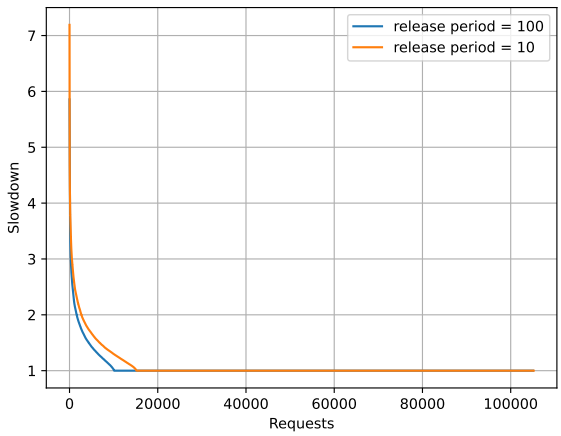


(b) Slowdown

Fig. 10.4: Comparison of processor usage and slowdown ratio when on-demand processors are not released and when released as soon as they are idle in a low load scenario.



(a) Processor Usage



(b) Slowdown

Fig. 10.5: Comparison of processor usage and slowdown ratio when on-demand processors are released with various periods in a high load scenario.

Table 10.3: Performance of DAL with and without release of on-demand processors under high load conditions.

Dispatch Policy	Missed Deadlines%	Dropped Jobs%
JSQ without Release	0.0028588	0
JSQ with Release Period = 1	0.00954372	0
JSQ with Release Period = 10	0.00475217	0
JSQ with Release Period = 100	0.00190391	0

Table 10.4: Performance of DAL with and without reserved processor provisioning.

Available Processors	Missed Deadlines%	Dropped Jobs%
No reserved processors	0.0243591	0
Four reserved processors	0.00954372	0

High Load Scenario: Under high loads, up to 10 on-demand processors were retained for 80 percent of the time in addition to the 4 reserved processors without release. When processors are released, only 3 additional processors are used for about 80 percent of the time. The slowdown ratio of the successful requests is 1 for more than 50 percent of the requests even when processors are released and can be seen in Fig. 10.3. Additionally, the percentage of deadlines missed is highly reduced (see Table 10.3).

No Reserved Processors: We also considered the scenario where an application is deployed only on-demand processors and no reserved processors are provisioned when the system load is set to 90 percent. The percentage of missed deadlines remains low while relatively higher compared to when 4 processors are reserved (Table 10.4).

Impact of Delayed Release: We evaluated DAL's performance by periodically releasing idle on-demand processors. We observed that releasing on-demand processors as soon as they become idle provides similar performance to not releasing them at all. Even with a delay in releasing them, the improvement is insignificant. A success rate of almost 99.99 percent is achieved in all cases, as shown in Table 10.3. We note that there is no statistically significant

Table 10.5: Performance of DAL when using using exact and mean execution time values to estimate response times.

Estimation Type	**Missed Deadlines%**	**Dropped Jobs%**
Distribution Mean with no on-demand Processors	19.6209	19.6047
Exact Value with no on-demand Processors	16.3863	16.3805
Distribution Mean with on-demand Processors	0.0058254	0
Exact Value with on-demand Processors	0.00232582	0

difference between JSQ with and without release, where the release period is set to 100. In terms of resource usage, Fig. 10.5 shows the different number of processors held by DAL during the simulation time. We observe that up to 13 processors are utilized for a relatively shorter duration of time even when processors are released as soon as they become idle. However, the average number of processors is significantly lower than the no release scenario while achieving almost identical performance in terms of successful completions. With respect to slowdown, we observe that the performance when processors are released periodically remains quite similar, with a slower release period having a slightly better value.

Impact of Execution Time Approximation: We evaluated the performance difference between using approximated execution times and exact execution times for scenarios where only reserved processors are utilized and for scenarios where on-demand processors are released as soon as they become idle. We found that in the former scenario, using approximated execution times results in approximately 6 percent higher deadline misses compared to using exact execution times, as seen in the first two rows of Table 10.5. However, this difference is almost insignificant when on-demand processors are utilized, with almost 99.99 percent of the jobs successfully meeting their deadlines.

Regarding processor usage, we observed that the use of approximation results in a slight increase in the average number of processors being used compared to the usage of exact execution times, as shown in Fig. 10.2. We also noticed an almost indiscernible difference in the slowdown ratio, with more than 85 percent of the jobs having a slowdown ratio of one for both cases. Furthermore, we found that using exact execution times of pending jobs to estimate

response times only marginally improves performance.

10.7 Conclusion

We considered the problem of dispatching and scheduling jobs that have variable execution times, arrival times as well as deadlines in an edge computing architecture. By assuming the availability of on-demand processors, we showed that DAL's dispatch-on-arrival policy along with per processor EDF scheduling policy, can achieve significantly better performance in terms of jobs that complete by their deadlines when jobs that miss deadlines are deleted from the queue. In terms of slowdown ratio, a significant percentage of the requests have a minimum achievable slowdown of 1. When on-demand processors are released periodically, both low and relatively high periods achieve similar performance in terms of missed deadlines although the slower release periods hold onto on-demand processors for a longer duration providing very little benefit.

Bibliography

[1] Shaik Mohammed Salman, Vaclav Struhar, Alessandro V Papadopoulos, Moris Behnam, and Thomas Nolte. Fogification of industrial robotic systems: Research challenges. In *Proceedings of the Workshop on Fog Computing and the IoT*, pages 41–45, 2019.

[2] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015.

[3] Miu-Ling Lam and Kit-Yung Lam. Path planning as a service ppaas: Cloud-based robotic path planning. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pages 1839–1844, 2014.

[4] Kaiyuan Eric Chen, Yafei Liang, Nikhil Jha, Jeffrey Ichnowski, Michael Danielczuk, Joseph Gonzalez, John Kubiatowicz, and Ken Goldberg. Fogros: An adaptive framework for automating fog robotics deployment. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 2035–2042, 2021.

[5] Zhihui Du, Ligang He, Yinong Chen, Yu Xiao, Peng Gao, and Tongzhou Wang. Robot cloud: Bridging the power of robotics and cloud computing. *Future Generation Computer Systems*, 74:337–348, 2017.

[6] Václav Struhár, Moris Behnam, Mohammad Ashjaei, and Alessandro V Papadopoulos. Real-time containers: A survey. In *2nd Workshop on*

Fog Computing and the IoT (Fog-IoT 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[7] Stefano Fiori, Luca Abeni, and Tommaso Cucinotta. Rt-kubernetes: Containerized real-time cloud computing. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, page 36–39, New York, NY, USA, 2022. Association for Computing Machinery.

[8] Akshitha Sriraman and Thomas F Wenisch. {μTune}:{Auto-Tuned} threading for {OLDI} microservices. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 177–194, 2018.

[9] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. Caladan: Mitigating interference at microsecond timescales. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 281–297, 2020.

[10] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high {CPU} efficiency for latency-sensitive datacenter workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 361–378, 2019.

[11] John P Lehoczky. Real-time queueing theory. In *17th IEEE Real-Time Systems Symposium*, pages 186–195. IEEE, 1996.

[12] John P Lehoczky. Using real-time queueing theory to control lateness in real-time systems. *ACM SIGMETRICS Performance Evaluation Review*, 25(1):158–168, 1997.

[13] 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.

[14] Ali Movaghar. On queueing with customer impatience until the beginning of service. *Queueing Systems*, 29(2):337–350, 1998.

[15] Mehdi Kargahi and Ali Movaghar. A method for performance analysis of earliest-deadline-first scheduling policy. *Journal of Supercomputing*, 37(2):197–222, 2006.

[16] Mahdieh Ahmadi, Morteza Golkarifard, Ali Movaghar, and Hamed Yousefi. Processor sharing queues with impatient customers and state-dependent rates. *IEEE/ACM Transactions on Networking*, 29(6):2467–2477, 2021.

[17] Miguel Alcon, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. Timing of autonomous driving software: Problem analysis and prospects for future solutions. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 267–280, 2020.

[18] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 286–295. IEEE Comput. Soc, 2-4 Dec. 1998.

[19] Chi-Sheng Shih and J.W.S. Liu. State-dependent deadline scheduling. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 3–14, 2002.

[20] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Joseph E. Gonzalez, and Ion Stoica. D3: A dynamic deadline-driven approach for building autonomous vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 453–471, New York, NY, USA, 2022. Association for Computing Machinery.

[21] Robert I. Davis and Liliana Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for real-time systems. *Leibniz Transactions on Embedded Systems*, 6(1):04:1–04:53, May 2019.

[22] Ala' Qadi, Steve Goddard, Jiangyang Huang, and Shane Farritor. Modelling computational requirements of mobile robotic systems using zones and processing windows. *Real-Time Systems*, 42:1–33, 2009.

[23] Shuang Wang, Xiaoping Li, Quan Z Sheng, Ruben Ruiz, Jinquan Zhang, and Amin Beheshti. Multi-queue request scheduling for profit maximization in iaas clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32(11):2838–2851, 2021.

[24] Jan Nouruzi-Pur, Jens Lambrecht, The Duy Nguyen, Axel Vick, and Jörg Krüger. Redundancy concepts for real-time cloud- and edge-based control of autonomous mobile robots. In *2022 IEEE 18th International Conference on Factory Communication Systems (WFCS)*, pages 1–8, 2022.

[25] Inho Cho, Ahmed Saeed, Joshua Fried, Seo Jin Park, Mohammad Alizadeh, and Adam Belay. Overload control for $\{\mu$s-scale$\}\{$RPCs$\}$ with breakwater. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 299–314, 2020.

[26] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[27] Luigi Biagiotti and Claudio Melchiorri. *Trajectory planning for automatic machines and robots*. Springer Science & Business Media, 2008.

[28] Paolo Pazzaglia, Claudio Mandrioli, Martina Maggio, and Anton Cervin. Dmac: Deadline-miss-aware control. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[29] 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.

[30] Nils Vreman, Claudio Mandrioli, and Anton Cervin. Deadline-miss-adaptive controller implementation for real-time control systems. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–26, 2022.

[31] Mehdi Kargahi and Ali Movaghar. Dynamic routing of real-time jobs among parallel edf queues: A performance study. *Computers & Electrical Engineering*, 36(5):835–849, 2010.

[32] Jiaying Meng, Haisheng Tan, Xiang-Yang Li, Zhenhua Han, and Bojie Li. Online deadline-aware task dispatching and scheduling in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1270–1286, 2019.

[33] Yiqin Gao, Guillaume Pallez, Yves Robert, and Frederic Vivien. Dynamic scheduling strategies for firm semi-periodic real-time tasks. *IEEE Transactions on Computers*, 72:55–68, 1 2023.

[34] Michael Mitzenmacher and Matteo Dell'Amico. The supermarket model with known and predicted service times. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2740–2751, 2022.

[35] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[36] Rusty O Baldwin, Nathaniel J Davis Iv, John E Kobza, and Scott F Midkiff. Real-time queueing theory: A tutorial presentation with an admission control application. *Queueing Systems*, 35(1):1–21, 2000.

[37] Alessandro Biondi and Youcheng Sun. On the ineffectiveness of 1/m-based interference bounds in the analysis of global edf and fifo scheduling. *Real-Time Systems*, 54(3):515–536, 2018.

Chapter 11

Paper D: Scheduling Firm Real-time Applications on the Edge with Single-bit Execution Time Prediction

Shaik Mohammed Salman, Van Lan Dao, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte.
IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC 2023)

Abstract

The edge computing paradigm brings the capabilities of the cloud such as on-demand resource availability to the edge for applications with low-latency and real-time requirements. While cloud-native load balancing and scheduling algorithms strive to improve performance metrics like mean response times, real-time systems, that govern physical systems, must satisfy deadline requirements. This paper explores the potential of an edge computing architecture that utilizes the on-demand availability of computational resources to satisfy firm real-time requirements for applications with stochastic execution and inter-arrival times. As it might be difficult to know precise execution times of individual jobs prior to completion, we consider an admission policy that relies on single-bit execution time predictions for dispatching. We evaluate its performance in terms of the number of jobs that complete by their deadlines via simulations. The results indicate that the prediction-based admission policy can achieve reasonable performance for the considered settings.

11.1 Introduction

Edge computing enables end-user applications to offload parts of their computations to achieve better response times and reduce energy consumption on local devices [1, 2, 3]. Some applications require the offloaded parts to be completed before a certain time for the results to be useful [4]. If the result has not been generated within this time, the computations (or jobs) can be abandoned. We refer to these types of applications as having firm real-time requirements [5]. To satisfy firm real-time requirements, it may be necessary to ensure that sufficient servers have been provisioned such that some performance metric, such as the average number of missed deadlines or the number of completed jobs, is below or above some threshold, respectively. In embedded settings, such resource provisioning is based on worst-case conditions such as worst-case execution times and minimum inter-arrival times [6]. While this approach can satisfy the timing requirements, it comes at the cost of inefficient resource usage. Provisioning based on average-case conditions, such as mean execution times and inter-arrival times, can achieve better resource usage, but we cannot guarantee performance deterministically [7, 8]. In this paper we investigate a potential approach to manage the conflicting requirements of efficiency and performance by considering an edge computing model where we provision a set of servers based on average-case conditions, while on-demand servers are provisioned to address transient and occasional worst-case conditions.

The scheduling strategies employed in such a computing model play a critical role in determining achievable performance. When aiming to minimize mean response times in single-server settings, scheduling strategies that are aware of the execution times of offloaded jobs, such as the shortest remaining processing time (SRPT), outperform strategies like first-in-first-out (FIFO) scheduling [9]. For multi-server settings where jobs are dispatched to specific servers upon arrival, the combination of SRPT and dispatching policies like join-shortest-queue (JSQ) are known to offer better performance [10]. However, in certain scenarios, execution times can only be known post-completion. To address this, several studies have proposed using predicted execution times in the absence of prior knowledge of true execution times [11, 12, 13, 14, 15]. Particularly in the context of queuing systems and aiming to improve the per-

formance metric of mean response times, Mitzenmacher [15] investigated the effectiveness of single-bit predictors that can determine whether a job's execution time is above or below a certain threshold. The study demonstrated that such predictors can offer benefits similar to those obtained with precise knowledge of execution times.

In addition to scheduling, load balancing or dispatching strategies can also affect performance. In terms of mean response times, load balancing techniques such as JSQ outperform strategies such as Round-Robin (RR) [16]. This benefit comes at the cost of increased overheads as JSQ policy requires the knowledge of pending jobs in each of the servers to identify the server with the least number of pending requests. For applications with response times in a few tens of milliseconds range, these overheads may be unacceptable and a low-complexity dispatching policy such as RR may be preferable. For applications with firm real-time requirements, in addition to removing jobs that can potentially miss their deadlines when already in the queue [4], it may be beneficial to admit only those jobs that can be estimated to finish by their deadlines while rejecting jobs that are most likely to miss their deadlines given the current pending jobs at the servers. This approach allows future jobs with possibly lower execution times and encountering reduced pending workload to be completed within deadlines, thereby improving the throughput.

Within this context, we consider applications with offloadable jobs whose execution times are given by a probability distribution and should be completed by a fixed relative deadline. Given the difficulty in knowing precise execution times of individual jobs, we assume the presence of a single-bit predictor[1] that can indicate if a job is a short job or a long a job and investigate the performance in terms of throughput, i.e, number of jobs that complete by their deadlines. The jobs are admitted or rejected on arrival based on an admission policy that estimates response time of the incoming job using the information from the single-bit predictor and dispatched following the JSQ or RR strategy with all admitted jobs scheduled in FIFO order. Furthermore, due to strong impossibility results in terms of competitive analysis, jobs must contain some, (possibly large) amount of slack [17, 18]. We include this in our work by setting the rel-

[1] Although single-bit prediction is coarse, it may be easier to realize them with better accuracy compared to fine-grained predictions.

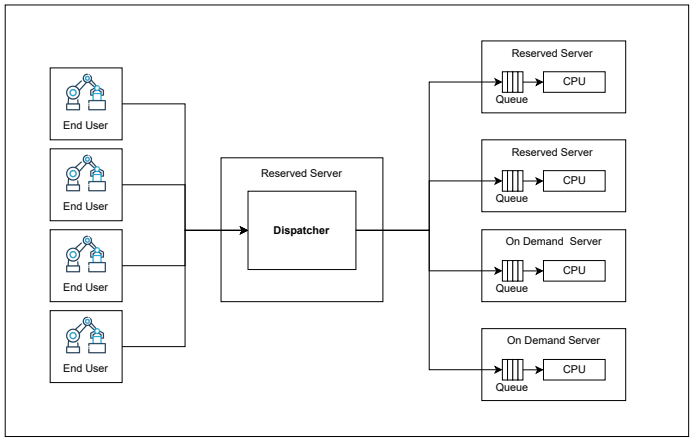


Fig. 11.1: System Architecture

ative deadline to be ten times the mean value of the execution time distribution. We use the value ten as we target applications with similar characteristics where the ratio between their worst-case execution times and the relative deadlines is large.

Concretely, we investigate the performance via simulations by considering exponential execution time distribution and Poisson arrivals. As we consider only the edge layer, we limit our simulations to non-asymptotic conditions. Additionally, we compare the performance of the prediction-based policy to a clairvoyant policy that has complete knowledge of the execution time of each pending as well as newly arriving job, as well as a policy that estimates the execution time of each job as the mean of its distribution. Our findings suggest that the prediction-based policy performs better than the mean-approximation policy, while being only slightly inferior to the clairvoyant policy when on-demand servers are available.

11.2 Background & Related Work

11.2.1 Edge Computing and On-Demand Servers

Edge computing architectures extend the concepts and benefits of cloud computing such as the provisioning of additional computing capabilities depending on the current workload for applications with latency and predictability requirements. Several auto-scaling strategies have been proposed that include vertical scaling where the amount of CPU time, (for example, in containerized deployments where a server is sharing a single CPU with other servers) allocated for a given application is increased or decreased, and horizontal scaling where the number of allocated servers is increased or decreased depending on average workload monitored over some time window [19]. In contrast, our edge-computing model is designed to address instantaneous load changes and assumes that all necessary setup for executing a job on on-demand servers is done during an initialization stage. We note that auto-scaling for sustained changes in workload can be easily incorporated into our model by changing the number of reserved servers. Close to the work presented in this paper, Wang et al. [20] provided a load balancer and core allocation strategy to minimize mean response times for non-preemptive FIFO by considering heterogeneous servers with reserved and on-demand servers within a cluster. Here the on-demand servers are utilized when the queue is full or when the waiting time exceeds the maximum waiting time. In our work, jobs are dispatched to on-demand servers based on the estimated response times. A distinguishing and reasonable assumption for edge computing is that the number of available computing nodes is much less compared to those of large data centers that make up the cloud infrastructure. As a consequence of this assumption, we restrict our analysis to small-sized clusters. In addition to this, we assume that the number of on-demand servers is also limited and known in advance. Fig. 11.1 depicts an example architecture with two reserved and two on-demand servers. Additionally, we assume that not all of the on-demand servers are available at all times for a specific application, as these on-demand servers may be shared among multiple applications. The availability is explicitly considered, and we model it as a Bernoulli distribution.

11.2.2 Dispatching Policies

In multi-server environments, dispatching policies determine the server on which an incoming job will be executed. In the online dispatching problem that we are considering, the dispatching decision is made when the job arrives at the dispatching server. Dispatching policies, such as joining the shortest queue and its variants, such as the shortest queue among k randomly chosen servers, require knowledge of the exact number of pending jobs in each of the considered servers [9]. Gathering this information may take a significant amount of time, depending on the number of servers and the network traffic [16]. As an alternative, policies such as round-robin are agnostic to pending jobs on the servers and dispatch incoming jobs to the servers in a repeating pattern. The advantage of policies such as round-robin is that they do not have the overhead associated with policies that require information about the pending workload. In most existing work, the objective of dispatching policies has been to minimize mean response times. Several additional policies, such as join-the-idle-queue and join-below-threshold, where servers notify the dispatchers when they are idle or have pending jobs less than a predefined value, have been proposed to balance the trade-off between overheads and response times [6, 16]. In this paper, we aim to evaluate the performance of the prediction-based admission policy in terms of achievable throughput and consider JSQ and RR as representative dispatching policies. We augment them with admission policies with the intuition that rejecting jobs that are unlikely to meet their deadlines can reduce the amount of time servers spend doing unuseful work.

11.2.3 Execution Time Predictions

Several works have investigated the possibility of improving the performance of algorithms with machine-learned advice or predictions including classical algorithms targeting problems such as online scheduling and load-balancing [11]. The evaluation of these prediction augmented algorithms has been done in terms of competitive analysis under accurate predictions and for possibly incorrect predictions [15, 21, 22, 6]. For the online scheduling problem, some of the authors have considered predicting parameters such as job execution times [6, 13] and permutation ordering of jobs [22]. Mitzenmacher [15] studied the

impact of single-bit predictors that can indicate if a job's execution time is above or below some threshold in the context of large-scale queuing systems for the performance metric of mean response times and showed that such predictors can provide benefits similar to those achievable with the knowledge of exact execution times for Poisson arrivals and certain execution time distributions. The analysis included the impact of incorrect predictions and highlighted the improvements achieved even with such incorrect predictions against the policy of choosing a queue with the least number of pending jobs. Similarly, they extended their analysis in [21] for the case where individual execution times were also predicted and showed via simulations that the benefits of the *supermarket model* in large distributed systems were retained if the predictions were *reasonably precise*. Based on the evaluations, they proposed the shortest queue selection and predicted shorted processing job first policy for use in actual systems, as it performed well in a diverse range of scenarios. In a similar context, Zhao et al.[23] extended the randomized multi-level feedback algorithm (RMLF) that makes no assumption on job execution times with predicted job execution times to minimize mean response times. Their experimental evaluation shows that the prediction-based algorithm achieves performance close to that of SRPT when the prediction error is small. If the error is large, their algorithm can achieve better performance than RMLF. An important requirement for such prediction-based enhancements is that the predictions should be learnable in practice. Keeping this in mind, we limit our attention in this paper to single-bit predictors that can identify jobs that can take a long time to execute and those that take a shorter duration. We use this information in our admission policy that estimates the response time of a new job and does a schedulability test using this response time. The work in our paper is inspired by the findings of these studies and is extended in the context of deadline constraints in terms of performance criteria while being restricted to single-bit predictions, FIFO order, and non-asymptotic conditions with probabilistic availability of on-demand servers.

***:** Scheduling Firm Real-time Tasks Gao et al. [4] proposed scheduling strategies for firm semi-periodic real-time tasks in single-server settings, where jobs are released periodically and have the same relative deadline, but execution

times have an arbitrary probability distribution. They investigated several optimization criteria, including the Deadline Miss Ratio (DMR). They introduced three new control parameters to decide at run-time whether to interrupt a job before its deadline. The parameters include (i) an upper bound on completion times, based on which a job is dropped if it is not completed by this time. This bound is a value between periodic inter-arrival time and relative deadline, (ii) an upper bound on job execution times, based on which jobs with execution times exceeding this value are rejected, and (iii) an upper bound on waiting time based on which a job that has waited until this bound will be dropped. In addition to this, they considered four admission policies which include (i) admitting all jobs, (ii) admitting jobs until a fixed number of jobs are in the queue, (iii) admitting jobs with some fixed probability and (iv) admitting jobs following a repeating pattern. Their evaluation shows that the key control parameter is the upper bound on the waiting time of each job achieving the best DMR. In comparison to this work, our work uses admission policies that estimate the response times based on the job execution time distribution and the number of pending jobs on a specific server while letting admitted jobs stay in the queue until their completion or until their deadline.

11.3 System Model

We now describe our system model and our assumptions in detail.

Specifying System Load: In queuing systems, it is essential that job arrival rates be less than departure rates to avoid queue build-up over time. For applications with execution time variability, a system designer has the option to consider inter-arrival times proportional to worst-case execution time or some value between the worst-case value and the mean of the execution time distribution. If the arrival times are proportional to worst-case values, the total number of jobs released over a fixed duration of time can be much less than when the arrival times are close to the mean values. Since we consider applications with firm real-time requirements, we define system load such that the arrival times are proportional to mean execution times rather than worst-case values. Our reasoning is based on the intuitive idea that even though executing

a large number of jobs can result in more of the jobs missing their deadlines, the number of jobs that complete by their deadlines can be larger than the number of jobs released when the arrival times are set to values proportional to worst-case execution times. This higher number of completed jobs can provide better functional performance than the scenario where no jobs miss their deadlines but only fewer jobs are released. Therefore, we model system load such that the average arrival rate is proportional to the mean of the execution time distribution and scales accordingly to the different number of reserved servers. Because we define system load based on average case parameters, there may be a situation where incoming jobs over a short duration of time take longer time to execute resulting in temporary overload. To manage this temporary overload, we consider the availability of on-demand computing resources within the edge layer.

Job Model: We assume that jobs have an execution time distribution with a known mean value μ. The exact execution time of a job remains unknown until its completion. All arriving jobs have a fixed relative deadline D. The jobs have a Poisson arrival process with a constant arrival rate λ. Whenever a job arrives, a dispatcher should decide if the job will be admitted or rejected. If admitted, each job remains in the server queue until completion or until its deadline. We assume that the fixed relative deadline is such that there is some amount of slack l with respect to the mean value μ. In this paper, we set l equal to ten times the mean μ. We set the system load proportional to the number of reserved servers. A job is said to be schedulable if its estimated response time is less than or equal to its relative deadline and unschedulable otherwise.

Server Model: We assume a cluster of homogeneous servers divided into a set of reserved servers R and a set of on-demand servers S. Each reserved server has its own queue and executes the jobs of a single application. Each admitted job is added to the queue of one of the servers on its arrival. Once assigned, the job stays in this queue until its completion. All admitted jobs are sequenced in FIFO order in each server queue and are executed non-preemptively. For the on-demand servers, each server can execute jobs for multiple applications and has a separate queue for each application. The server is considered

available for a specific application if it meets one of the following criteria: (i) it is idle, (ii) it is executing jobs of the same application and has no pending jobs from higher-priority applications, or (iii) it is executing jobs of a lower-priority application. If the server does not meet any of these criteria, it is considered unavailable for the application. In our simulations, we model the availability of the on-demand servers using a Bernoulli distribution. Although this availability model is simple, it enables a straight-forward quantitative comparison of the different admission and dispatching policy combinations.

11.4 Admission Policies and Dispatching

In our on-arrival dispatching model, an admission policy determines whether a job should be accepted or rejected. In this work, we consider three admission policies that estimate the response time of an incoming job by taking into account the pending jobs on a given server (see Section 11.4.1). Admission or rejection can be done in two steps. In the first step, the dispatcher looks for a server within the set of reserved servers. If it fails to admit a job onto a reserved server, it searches for an available on-demand server and checks for schedulability. A job is admitted if the policy considers it to be schedulable on a reserved server, and then it is sent to that server by the dispatcher. If a job is unschedulable on a reserved server, the dispatcher tries to find an available on-demand server and tests the schedulability of the job on this on-demand server. It dispatches the job to it if it is schedulable. Otherwise, the job is rejected. Algorithm 1 describes our dispatching process, which takes as input the identifiers of the reserved servers and on-demand servers, the dispatching policy, the relative deadline, and the specific admission policy.

If the configured dispatching policy is JSQ, the dispatcher selects the server with the shortest queue among the reserved servers. If the dispatching policy is RR it cyclically selects a server. When a new job arrives and a reserved server has been identified, the dispatcher uses one of the response time estimators and checks if the estimated response time is less than or equal to the relative deadline. If so, the job is sent to the identified server (lines 9-11). If the response time estimate is not within the deadline, the dispatcher identifies the subset of available on-demand servers and selects a server according to the configured

Algorithm 1 Dispatcher

Input: Incoming job J, set of reserved servers R and set of on demand servers S, dispatching policy P, deadline D, response time estimation policy A
Output: The id of server if job schedulable, $NULL$ otherwise.
function GETBESTSERVER(J, R, S, P, D)
    if $P == JSQ$ **then**
        $id \leftarrow get\_shortest\_queue\_server(R)$
    else if $P == RR$ **then**
        $id \leftarrow get\_next\_rr\_server(R)$
    end if
    $f_i \leftarrow get\_estimated\_response\_time(id, J, A)$
    if $f_i \leq D$ **then**
        return id ▷ job deemed to be schedulable
    else if $f_i > D$ **then**
        $S_x \leftarrow get\_available\_on\_demand\_servers(S)$
        if $P == JSQ$ **then**
            $id \leftarrow get\_shortest\_queue\_server(S_x)$
        else if $P == RR$ **then**
            $id \leftarrow get\_next\_rr\_server(S_x)$
        end if
        $f_i \leftarrow get\_estimated\_response\_time(id, J, A)$
        if $f_i \leq D$ **then**
            return id ▷ job deemed to be schedulable
        else
            return $NULL$ ▷ job deemed to be unschedulable
        end if
    end if
end function

policy. The dispatcher then uses one of the response time estimators and checks if the estimated response is less than or equal to the relative deadline. If so, the job is sent to the identified on-demand server (lines 19-21). If the estimated response time is not within the deadline, the job is rejected.

11.4.1 Response Time Estimation

We now describe the response time estimation policies used to admit or reject the jobs. We consider three policies based on how the job execution times are considered, (i) mean-approximation policy, (ii) clairvoyant policy, and (iii) single-bit prediction policy.

Mean-approximation policy: In this policy, we use mean μ of the execution time distribution to estimate the response time f_i on the selected server i. If the number of pending jobs on this server is given by N_i, the estimated response time is given by

$$f_i = (N_i + 1) \cdot \mu. \tag{11.1}$$

We consider the mean-approximation policy because of its low computational overhead.

Clairvoyant policy: In this policy, we assume the knowledge of exact execution times. The response time f_i on any server i is given by

$$f_i = x_j + \sum_{k=0}^{N_i} x_k, \tag{11.2}$$

where x_k is the exact execution time of each job k assigned to server i and x_j is the execution time of the newly arrived job.

Single-bit prediction policy : In this policy, we assume that there exists a predictor which can indicate if a job is a short job or a long job. A job is said to be a short job if its true execution time is less than μ and long otherwise. The response time f_i of a job on any server i where N_i^s is the number of pending

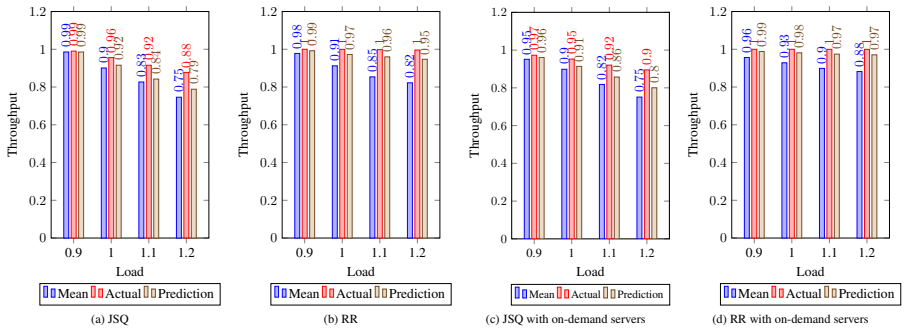


Fig. 11.2: Throughput of various admission tests and load for exponentially distributed service times for (a) JSQ, (b) RR, (c) JSQ with on-demand servers, and (d) RR with on-demand servers.

jobs classified as short and N_i^l is the number of pending jobs classified as long is given by

$$f_i = (N_i^s \cdot \mu_s) + (N_i^l \cdot \mu_l) + (\tau \cdot \mu_s + (1 - \tau) \cdot \mu_l), \tag{11.3}$$

where μ_s and μ_l are specified by the designer and τ is the output of predictor indicating if the new job is a short job or a long job. We assume that a server can identify both N_i^s and N_i^l and make this information available to the dispatcher.

For all of the estimation methods, the admission test returns true if the following condition is satisfied:

$$f_i \leq D. \tag{11.4}$$

11.5 Evaluation

We use simulation to evaluate the performance of the prediction-based policy in terms of throughput. We consider throughput as the ratio of jobs that completed before or at their deadlines and the total number of jobs that arrived during the simulation interval. We set the simulation interval to ten thousand time units and take the average of the measured throughput for ten simulation runs. We

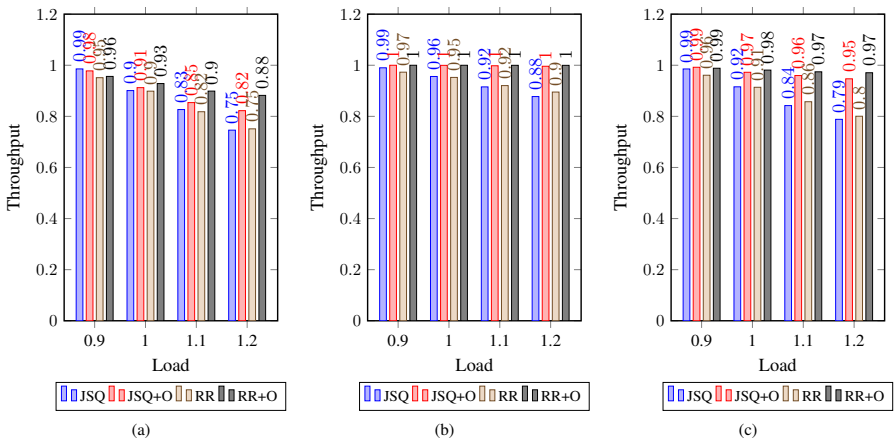


Fig. 11.3: Throughput under various load conditions when using on-demand servers in addition to reserved servers for admission policies with (a) mean (b) exact execution time (c) Single-bit job type prediction

generated the inter-arrival times of the jobs and the execution time distributions using the exponential distribution class of the C++ library. We set the mean of execution time distribution μ equal to ten and the arrival rate as proportional to the inverse of μ. Each generated job is assigned a deadline equal to ten times the mean of the distribution. We set μ_s equal to five and μ_l equal to fifteen. The number of reserved servers was varied between two, four, and eight but we only present the results for the scenario where the number of servers was set to eight. Similarly, we set the number of on-demand servers equal to the number of reserved servers. We make the assumption that there are no overheads involved in obtaining the state of the queues from the servers.

11.5.1 Performance of Prediction-Based Admission Policy

We compare the performance of the admission policy that uses the information provided by the single-bit predictor against a baseline solution that uses the exact execution time information and another solution that uses the mean of the service time distribution to estimate the response times of newly arriving jobs

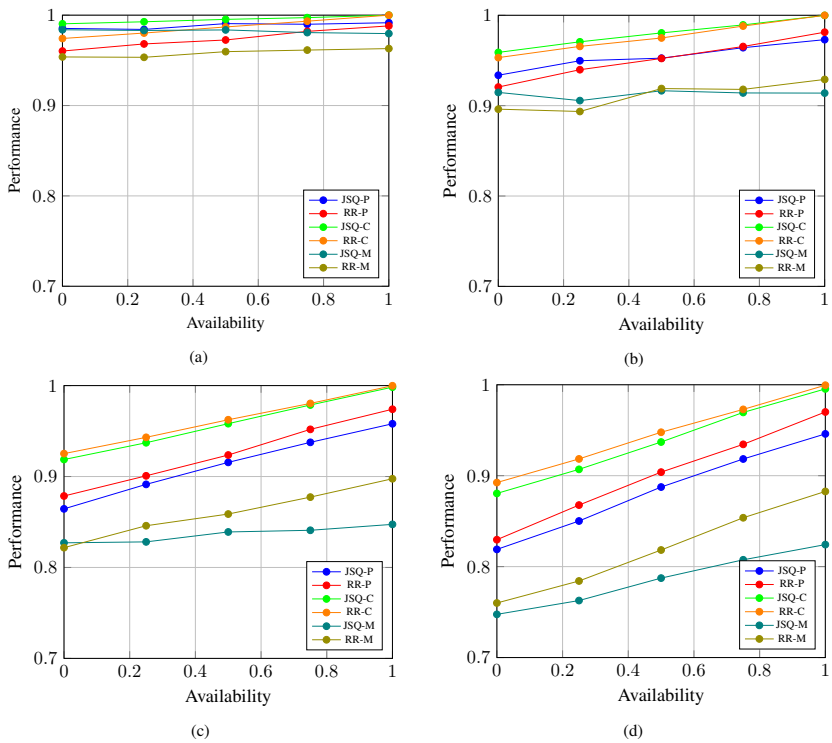


Fig. 11.4: Throughput for varying availability probability of on-demand servers when (a) Load = 0.9, (b) Load = 1, (c) Load = 1.1, and (d) Load =1.2

before deciding to admit or reject the jobs. We evaluate this performance for varying loads, dispatching policies, and availability of on-demand servers.

As seen in Fig. 11.2 when only using the reserved servers, the difference between the throughput achieved by the prediction-based policy and the clairvoyant policy becomes more apparent as the load is increased. This behavior is observed for both dispatching policies. When on-demand servers are always available, the observations remain consistent. The benefits of the prediction-based policy are more evident when compared to the mean-approximation policy at higher loads and always available on-demand servers.

> Prediction-based admission policy provides better throughput performance compared to mean-approximation policy while the clairvoyant policy provides the best throughput among all the considered policies.

11.5.2 Performance of Dispatching Policies

Our evaluation indicates that both JSQ and RR have similar throughput performance for loads greater than one when only using reserved servers for identical admission policies. This indicates that the dispatching policy has very little impact on the throughput compared to the impact of the response time estimation policies. When on-demand servers are considered to be always available, the performance difference remains negligible as seen in Fig. 11.2 and Fig. 11.3.

> Both JSQ and RR have similar performance when using identical admission policies on reserved servers. When on-demand servers are always available, RR achieves similar average throughput compared to JSQ.

11.5.3 Performance Impact of Availability of On-Demand Servers

We investigate the impact of the availability of on-demand servers on the achievable throughput for different admission policies and dispatching combinations. As seen in Fig. 11.3 When the load is close to 0.9, the impact of on-demand server availability is negligible for all the admission and dispatching policies. However, when the load is increased, the availability of on-demand servers provides considerable improvement in throughput. Clairvoyant policy

dominates the performance for all of the availability values. Increasing availability results in improved performance for all of the admission policies as seen in Fig. 11.4. Additionally, combining the prediction-based policy with either of the dispatching policies outperforms the mean-approximation policy while closely following the clairvoyant policy.

> Increased availability of on-demand servers improves throughput significantly under overload conditions for all admission policies compared to using only reserved servers.

11.6 Conclusion

We have studied the performance of a single-bit prediction based admission policy for the problem of online dispatching and scheduling of jobs with stochastic execution and inter-arrival times, along with deadline constraints for firm real-time systems in a multi-server edge computing environment. Using simulations, we evaluated and compared the performance of the prediction based policy against the mean-approximation and the clairvoyant admission policy with two well-known dispatching strategies, JSQ and RR. We have also considered an architecture that provides access to on-demand servers in addition to a set of reserved servers. Our results indicate that the achievable throughput is primarily influenced by the estimation accuracy of the admission policy and availability of on-demand servers, rather than the dispatching policy. In addition, the single-bit prediction policy outperformed mean-approximation policy while falling short of the performance of the clairvoyant policy.

Acknowledgement

The research leading to these results has received funding from the Knowledge Foundation (KKS), under the projects FIESTA (Project No. 20190034) and SACSys (Project No. 20190021), and under the Swedish Research Council (VR), under the project PSI (Project No. 2020-05094).

Bibliography

[1] Josip Zilic, Atakan Aral, and Ivona Brandic. Efpo: Energy efficient and failure predictive edge offloading. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 165–175, 2019.

[2] Josip Zilic, Vincenzo De Maio, Atakan Aral, and Ivona Brandic. Edge offloading for microservice architectures. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '22, page 1–6, New York, NY, USA, 2022. Association for Computing Machinery.

[3] Matthijs Jansen, Auday Al-Dulaimy, Alessandro V. Papadopoulos, Animesh Trivedi, and Alexandru Iosup. The spec-rg reference architecture for the edge continuum, 2022.

[4] Yiqin Gao, Guillaume Pallez, Yves Robert, and Frederic Vivien. Dynamic scheduling strategies for firm semi-periodic real-time tasks. *IEEE Transactions on Computers*, 72:55–68, 1 2023.

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

[6] Yecheng Zhao, Runzhi Zhou, and Haibo Zeng. Design optimization for real-time systems with sustainable schedulability analysis. *Real-Time Systems*, 58(3):275–312, 2022.

[7] 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*, pages 164–173. IEEE, 1995.

[8] J.L. Diaz, D.F. Garcia, Kanghee Kim, Chang-Gun Lee, L. Lo Bello, J.M. Lopez, Sang Lyul Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 289–300, 2002.

[9] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[10] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. Load balancing guardrails: Keeping your heavy traffic on the road to low response times (invited paper). In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 10, New York, NY, USA, 2021. Association for Computing Machinery.

[11] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

[12] Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 1070–1080, New York, NY, USA, 2021. Association for Computing Machinery.

[13] Ziv Scully, Isaac Grosof, and Michael Mitzenmacher. Uniform bounds for scheduling with job size estimates. *arXiv preprint arXiv:2110.00633*, 2021.

[14] Maryam Akbari-Moghaddam and Douglas G Down. Seh: Size estimate hedging scheduling of queues. *ACM Transactions on Modeling and Computer Simulation*, 2023.

[15] Michael Mitzenmacher. Queues with small advice. In *SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, pages 1–12. SIAM, 2021.

[16] Xingyu Zhou, Fei Wu, Jian Tan, Yin Sun, and Ness Shroff. Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–30, 2017.

[17] Franziska Eberle, Nicole Megow, and Kevin Schewior. Optimally handling commitment issues in online throughput maximization. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[18] Franziska Eberle, Nicole Megow, and Kevin Schewior. Online throughput maximization on unrelated machines: Commitment is no burden. *ACM Transactions on Algorithms*, 12 2022.

[19] Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Comput. Surv.*, 51(4), jul 2018.

[20] Shuang Wang, Xiaoping Li, Quan Z Sheng, Ruben Ruiz, Jinquan Zhang, and Amin Beheshti. Multi-queue request scheduling for profit maximization in iaas clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32(11):2838–2851, 2021.

[21] Michael Mitzenmacher and Matteo Dell'Amico. The supermarket model with known and predicted service times. *IEEE Transactions on Parallel and Distributed Systems*, 33:2740–2751, 11 2022.

[22] Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, page 357–368, New York, NY, USA, 2022. Association for Computing Machinery.

[23] Tianming Zhao, Wei Li, and Albert Y. Zomaya. Real-time scheduling with predictions. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 331–343, 2022.

Chapter 12

Paper E: Taming Tardiness on Parallel Machines: Online Scheduling with Limited Job Information

Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte.
MRTC Report, MDU 2024.

Abstract

We consider the problem of scheduling n jobs on $m \geq 2$ parallel machines in online settings with the objective of minimizing total tardiness. Since no bounded competitive algorithms exist to minimize the general problem of weighted total tardiness of the form $\sum w_j T_j$, we consider an objective of the form $\sum w_j(T_j + d_j)$, where w_j, T_j, and d_j are the weight, tardiness, and deadline of each job, respectively and develop competitive algorithms dependent on jobs' processing times.

12.1 Introduction

We consider an online scheduling problem where n jobs arrive at arbitrary times and should be completed before or close to their deadlines on a set of m parallel machines. At any given time, a job can be scheduled on at most one machine. All jobs that arrive must run to completion. They cannot be aborted or rejected. Preemption and possibly migration are allowed. A natural objective for this problem is to minimize tardiness (T_j), which is the difference between the completion time of a job and its deadline if the job completes after its deadline, and zero otherwise. A generalization of this objective is to minimize the weighted total tardiness. In this context, each job has a deadline constraint and a weight. In Graham's $\alpha|\beta|\gamma$ notation, where α is the machine environment, β describes the job characteristics and constraints, and γ specifies the objective function, these settings can be described as the online variants of $P|r_j, p_j, d_j, pmtn|\sum w_j T_j$ and $R|r_j, p_j, d_j, pmtn|\sum w_j T_j$ where r_j, p_j and d_j denote release time, processing time and deadline of each job j. P and R represent identical and unrelated machines' settings, respectively.

This problem is well-known to be NP-hard even on a single machine. Due to the inherent difficulty in developing competitive algorithms for this problem, we consider a modified tardiness objective of the form $\sum w_j(T_j + d_j)$. This modified objective was introduced in the offline version of the problem with unit weights in which all jobs have a common deadline by Kovalyov and Werner [1]. Kolliopoulos and Steiner [2] considered the general version of the problem with arbitrary weights and showed a reduction to the problem of finding an approximate solution to the problem of weighted total completion time. Their result showed that any ρ-approximation algorithm for the problem of minimizing total weighted completion time was an $(\rho + 1)$-approximation algorithm for the problem of minimizing weighted modified total tardiness. Liu et al. [3] extended this idea to online settings in addition to an availability constraint on the machines. They provided $O(1)$-competitive algorithms for clairvoyant scenarios. Specifically, for the single-machine version of the problem with weights, they showed a 2-competitive lower bound and a 3-competitive algorithm as an upper bound. For the multiple-machine version of the unit weight problem, they provided a lower bound of 1.309 and a 3-competitive algorithm.

Kolliopoulos and Steiner [2] considered a stochastic version of the problem where the processing times of the jobs are assumed to be random variables with known distributions.

As clairvoyant information is difficult to obtain in several practical applications, this paper examines semi-clairvoyant and prediction-clairvoyant scenarios where the processing time information of a job is limited at its release time. In this context, *limited* refers to the fact that a scheduling algorithm utilizes a proxy that either depends on the knowledge of the range or on an estimation of the job's processing time instead of its true processing time. For these scenarios, we develop competitive algorithms addressing the modified total tardiness objective. Specifically, we make the following contributions.

- In prediction-clairvoyant settings: an $O(\mu \log \hat{P})$-competitive algorithm without migration for parallel identical machines with unit weights.
- In semi-clairvoyant settings: an $O(\log P)$-competitive algorithm for parallel identical machines with unit weights.
- In speed-prediction settings: an $O(\mu)$-competitive algorithm for parallel unrelated machines with general weights and clairvoyant processing times.

12.2 Preliminaries

Each job j arrives at a time r_j and has a deadline d_j such that $r_j \leq d_j$. The job has an actual processing time p_j. Upon arrival, either an estimated processing time $\tilde{p_j}$ or its job class l_j is available. Let C_j denote the time at which a job completes its execution. Reusing terminology and notation from [4], we let $\mu_1 = \max_j \frac{p_j}{\tilde{p_j}}$ be maximum underestimation error among all the arriving jobs. Similarly, we let $\mu_2 = \max_j \frac{\tilde{p_j}}{p_j}$ be maximum overestimation error among all the arriving jobs and $\mu = \mu_1 \cdot \mu_2$ be the distortion parameter.

We let P be the ratio between the maximal actual processing time and the minimal actual processing time among the jobs, i.e., $P = \frac{\max_j p_j}{\min_j p_j}$. Additionally, we let $\hat{P}$ be the ratio between the maximal estimated processing time and

the minimal estimated time among the jobs, i.e., $\hat{P} = \frac{\max_j \tilde{p_j}}{\min_j \tilde{p_j}}$. Furthermore, we use the following definitions.

Definition 4 (Predicted job class)**.** *We define the predicted class $\tilde{\ell_j}$ of a job j, as an integer k such that $\tilde{p_j} \in [2^k, 2^{k+1}]$.*

Definition 5 (Job class)**.** *We define the class ℓ_j of a job j, as an integer k such that $p_j \in [2^k, 2^{k+1}]$.*

Definition 6 (Tardiness)**.** *We define tardiness T_j of a job j as $\max(C_j - d_j, 0)$.*

Definition 7 (Modified Tardiness)**.** *We define modified tardiness $\tilde{T_j}$ of a job j as $T_j + d_j$.*

Definition 8 (Total Modified Tardiness)**.** *We define total modified tardiness as $\sum \tilde{T_j}$.*

Definition 9 (Total Completion Time)**.** *We define total completion time as $\sum C_j$.*

Definition 10 (Total Flow Time)**.** *We define total flow time as $\sum C_j - r_j$.*

12.3 Prediction-Clairvoyant Scheduling on Parallel Machines

In this section, we focus on our first contribution, which addresses the problem of minimizing the total modified tardiness on parallel identical machines utilizing predicted job processing times. For this, we reuse the prediction-based algorithm by Azar et al. [4] that was developed for the problem of minimizing the total flow time with a competitive ratio of $O(\mu \log \hat{P})$. This algorithm satisfies the consistency and robustness properties [5] desired from prediction-based online algorithms. When the value of the distortion parameter is close to one due to high-quality predictions, the algorithm has a competitive ratio $O(\log \hat{P})$, which matches the lower bound for clairvoyant settings, satisfying the consistency property. We prove that this algorithm has an identical competitive ratio for the objective of modified total tardiness. Formally, we prove the following result.

Theorem 12.3.1. *Algorithm 1 is $O(\mu \log \hat{P})$-competitive for minimizing modified total tardiness on parallel identical machines with predicted job processing times.*

12.3.1 Algorithm

Algorithm 2 utilizes job classes based on the predicted processing time to prioritize jobs. When a new job arrives, it is placed at the top of the stack of an available machine or a machine currently processing a job of a higher class. Otherwise, it is added to a central queue. It assigns higher priority to jobs with lower classes. Higher-class jobs wait in the central queue until a machine becomes available or a lower-class job completes processing. Preempted jobs are returned to the machine stack on which they were preempted and resumed later on the same machine.

Algorithm 2 Distortion Oblivious Non-Migrative Scheduling Algorithm

function UPONJOBRELEASE(j)
    if exists an idle machine or a machine that currently processes a job of a class higher than $\hat{\ell}_j$ **then**
        Insert j to the top of that machine stack.
    else
        Insert j to the pool.
    end if
end function
function UPONJOBCOMPLETION(j)
    Denote by m_j the machine j was processed on.
    Pop j from the stack of m_j, and let j' be the next job in that stack.
    if the job with lowest class in the pool j'' has class strictly less than that of j' or j' does not exist **then**
        Remove j'' from the pool and insert it to the top of the stack of m_j.
    end if
end function

12.3.2 Analysis

Algorithm 2 was originally developed to minimize total flow time, and we reuse it without modifications for the objective of total modified tardiness. To show that this algorithm indeed works for our objective, we use a proof method similar to the one used in [2] and [3]. We first utilize the result that reducing the problem of the total completion time minimization to the problem of total flow time minimization increases the competitive ratio by a constant multiplicative factor of 2. We then utilize the result that reducing the tardiness minimization problem to the problem of total completion time minimization problem increases the competitive ratio by an additive constant 1. Lastly, we plug in the asymptotic bound of Algorithm 2. Formally, we need the following lemmas to prove Theorem 12.3.1.

Lemma 12.3.2. *(Theorem 1 from [2]) Consider a member $\alpha_0|\beta_0|\sum_j w_jC_j$ of the family of scheduling problems $\alpha|\beta|\sum_j w_jC_j$ for which there is a γ-approximation algorithm. Then the same algorithm achieves a $(\gamma+1)$-approximation for the problem $\alpha_0|\beta_0|\sum_j w_j(T_j+d_j)$.*

Proof. See [2] for a comprehensive proof. □

As two of the algorithms we consider in this paper address the problem of total flow time, we use the results from [6] that provide a reduction from the problem of minimizing total completion time to the problem of minimizing total flow time. Combining Theorem 7.3 and lemma 7.6 from [6], we have the following result.

Lemma 12.3.3. *Consider the problem of minimizing the total weighted flow time, for which a s-speed, c-competitive algorithm exists. The same algorithm is $2cs$-competitive for the problem of minimizing total weighted completion time.*

Proof. See chapter 7 in [6] for a comprehensive proof. □

The next lemma states the competitiveness of Algorithm 2 for the objective of total flow time.

Lemma 12.3.4. *(Theorem 2 from [4]) Algorithm 2 is $O(\mu \log \hat{P})$-competitive for inputs with distortion μ for the problem of minimizing total flow time on parallel identical machines.*

Proof. See [4] for a comprehensive proof. □

Proof of Theorem 12.3.1 With the preceding lemmas in place, we are now ready to prove Theorem 1. Let $C_j(alg)$ and $T_j(alg)$ be the completion times and tardiness of n jobs, respectively, due to the scheduling policy of Algorithm 1. Let $C_j(opt)$ and $T_j(opt)$ be the completion times and tardiness due to an optimal algorithm for the same instance of jobs.

By definition, we have the following equality.

$$\sum (T_j + d_j) = \sum \max(C_j, d_j) \tag{12.1}$$

The left-hand side of Eq. (12.1) represents the total modified tardiness as the sum of tardiness of individual jobs and their respective deadlines. This is equivalent to the sum of the maximum between the completion time C_j and the deadline d_j of each job j.

Splitting the sum of the right-hand side into two separate sums, we have the following inequality.

$$\sum \max(C_j(alg), d_j) \leq \sum C_j(alg) + \sum d_j \tag{12.2}$$

Similarly, we have the following lower bound.

$$\sum \max(C_j(opt), d_j) \leq \sum C_j(opt) + \sum d_j \tag{12.3}$$

From the definition of competitive ratio, we have the following inequality bounding the total completion time.

$$\sum C_j(alg) + \sum d_j \leq c \cdot \sum C_j(opt) + \sum d_j \tag{12.4}$$

Since Algorithm 2 minimizes total flow time, from Lemma 12.3.3, we know that any algorithm that is c-competitive for minimizing total flow time bounds the total completion time by a multiplicative factor of 2c of the optimal completion time. Using this fact, we have the following inequality.

$$\sum \max(C_j(alg), d_j) \leq 2 \cdot c \cdot \sum C_j(opt) + \sum d_j \tag{12.5}$$

Dividing the above equation with the respective lower bounds, we get

$$\frac{\sum \max(C_j(alg), d_j)}{\sum \max(C_j(opt), d_j)} \leq \frac{2 \cdot c \cdot \sum C_j(opt) + \sum d_j}{\sum C_j(opt) + \sum d_j} \tag{12.6}$$

Splitting the right-hand side of the above inequality into two separate terms, we get

$$\frac{2 \cdot c \cdot \sum C_j(opt)}{\sum C_j(opt) + \sum d_j} \leq 2 \cdot c \tag{12.7}$$

Similarly,

$$\frac{\sum d_j}{\sum C_j(opt) + \sum d_j} \leq 1 \tag{12.8}$$

Combining Eq. (12.7) and Eq. (12.8) and rearranging the terms in equation Eq. (12.6), we get

$$\sum \max(C_j(alg), d_j) \leq (2 \cdot c + 1) \cdot \sum \max(C_j(opt), d_j) \tag{12.9}$$

From Lemma 12.3.4, we can replace the constant c with asymptotic bound $O(\mu \log \hat{P})$ of Algorithm 2. Ignoring the constant factors, we have the following inequality.

$$\sum \max(C_j(alg), d_j) \leq O(\mu \log \hat{P}) \cdot \sum \max(C_j(opt), d_j) \tag{12.10}$$

Using the definition from Eq. (12.1) and rewriting the above equation, we get

$$\sum (T_j(alg) + d_j) \leq O(\mu \log \hat{P}) \cdot \sum (T_j(opt) + d_j) \tag{12.11}$$

The claim follows. □

12.4 Semi-Clairvoyant Scheduling on Parallel Machines

This section focuses on our contribution related to the semi-clairvoyant settings. This is an alternative approach to address the problem of the unavailability of precise processing times at the time of their release. Here, a scheduler utilizes the approximate knowledge of the processing time of a job in terms of its class to determine the order in which the jobs are processed instead of its actual processing time. A job's class is an integer value that identifies the processing time range of the job. While one can assume that a job's class is known precisely, it is possible to consider the scenario in which a job's class is also predicted. For each of these scenarios, the lowest-class-first (LCF) algorithm developed by Bechetti et al. [7] has been proven to be $O(\log P)$ and $O(\mu \log \hat{P})$ competitive for the total flow time objective, respectively. These bounds match proven lower bounds for the total flow time objective. We reuse this algorithm to minimize modified total tardiness. Formally, we prove the following results.

Theorem 12.4.1. *Algorithm 3 is $O(\log P)$-competitive for the problem of minimizing modified total tardiness on parallel identical machines in semi-clairvoyant settings.*

Theorem 12.4.2. *Algorithm 3 is $O(\mu \log \hat{P})$-competitive for the problem of minimizing modified total tardiness on parallel identical machines in semi-clairvoyant settings with job class predictions.*

12.4.1 Algorithm

Algorithm 3 assumes that each job reveals its actual (predicted) class on its arrival. When a new job arrives, it is assigned to an available machine or a machine currently processing a job of a higher class. Otherwise, it is added to a central queue. This approach assigns higher priority to jobs with lower classes, while higher-class jobs wait in the central queue until a machine becomes available or a lower-class job completes processing.

Algorithm 3 Lowest Class First Scheduling Algorithm

```
1: function UponJobRelease(j)
2:     if exists an idle machine or a machine that currently processes a job k
       of a class higher than ℓ_j(ℓ̂_j) then
3:         Preempt k and insert it into the pool.
4:         Run j.
5:     else
6:         Insert j into the pool.
7:     end if
8: end function
9: function UponJobCompletion(j)
10:    Denote by m_j the machine j was processed on.
11:    if there exists a job in the pool j'' then
12:        Remove j'' with the lowest class from the pool and run on m_j.
13:    end if
14: end function
```

12.4.2 Analysis

To show that Algorithm 3 is competitive for the objective of modified tardiness minimization, our analysis follows an identical approach to our proof of Theorem 12.3.1. Specifically, we utilize the reduction in Lemma 12.3.3 that bounds the total completion time by a factor of 2c of the optimal total completion time when using an algorithm designed to minimize total flow time. We then utilize the reduction in Lemma 12.3.2 that bounds the total modified tardiness by a factor of $c + 1$. Finally, we plug in the asymptotic upper bound for flow time minimization. Formally, to prove Theorem 12.4.1, we need an additional result stated in Lemma 12.4.3.

Lemma 12.4.3. *(Theorem 15 from [7]) Algorithm 3 is $O(\log P)$-competitive for the problem of minimizing total flow time on parallel identical machines.*

Proof. See [7] and [8] for a comprehensive proof. □

Proof of Theorem 12.4.1 Let $C_j(alg)$ and $T_j(alg)$ be the completion times and tardiness of n jobs, respectively, due to the scheduling policy of Algorithm 3.

Let $C_j(opt)$ and $T_j(opt)$ be the completion times and tardiness due to an optimal algorithm for the same instance of jobs.

The left-hand side of Eq. (12.1) represents the total modified tardiness as the sum of tardiness of individual jobs and their respective deadlines. This is equivalent to the sum of the maximum between the completion time $C_j(alg)$ and the deadline d_j of each job j.

Splitting the sum of the right-hand side of Eq. (12.1) into two separate sums, we have the following inequality.

$$\sum \max(C_j(alg), d_j) \leq \sum C_j(alg) + \sum d_j \tag{12.12}$$

Similarly, we have the following lower bound.

$$\sum \max(C_j(opt), d_j) \leq \sum C_j(opt) + \sum d_j \tag{12.13}$$

From the definition of competitive ratio, we have the following inequality bounding the total completion time.

$$\sum C_j(alg) + \sum d_j \leq c \cdot \sum C_j(opt) + \sum d_j \tag{12.14}$$

Since Algorithm 3 minimizes total flow time, from Lemma 12.3.3, we know that any algorithm that is c-competitive for minimizing total flow time bounds the total completion time by a multiplicative factor of 2c of the optimal completion time. Using this fact, we have the following inequality.

$$\sum \max(C_j(alg), d_j) \leq 2 \cdot c \cdot \sum C_j(opt) + \sum d_j \tag{12.15}$$

Dividing the above equation with the respective lower bounds, we get

$$\frac{\sum \max(C_j(alg), d_j)}{\sum \max(C_j(opt), d_j)} \leq \frac{2 \cdot c \cdot \sum C_j(opt) + \sum d_j}{\sum C_j(opt) + \sum d_j} \tag{12.16}$$

Splitting the right-hand side of the above inequality into two separate terms, we get

$$\frac{2 \cdot c \cdot \sum C_j(opt)}{\sum C_j(opt) + \sum d_j} \leq 2 \cdot c \tag{12.17}$$

Similarly,

$$\frac{\sum d_j}{\sum C_j(opt) + \sum d_j} \leq 1 \tag{12.18}$$

Combining Eq. (12.17) and Eq. (12.18) and rearranging the terms in equation Eq. (12.6), we get

$$\sum \max(C_j(alg), d_j) \leq (2 \cdot c + 1) \cdot \sum \max(C_j(opt), d_j) \tag{12.19}$$

Using the result from Lemma 12.4.3 and ignoring the constant factors, we have the following inequality.

$$\sum \max(C_j(alg), d_j) \leq O(\mu \log P) \cdot \sum \max(C_j(opt), d_j) \tag{12.20}$$

Using the definition from Eq. (12.1) and rewriting the above equation, we get

$$\sum (T_j(alg) + d_j) \leq O(\mu \log P) \cdot \sum (T_j(opt) + d_j) \tag{12.21}$$

The claim follows. □

To prove Theorem 12.4.2, we need the following result.

Lemma 12.4.4. *(Theorem 14 from [4]) Algorithm 3 is $O(\mu \log \hat{P})$-competitive for inputs with distortion μ for the problem of minimizing total flow time on parallel identical machines with predicted job classes.*

Proof. See [4] for a comprehensive proof. □

Proof of Theorem 12.4.2 Our proof is identical to the proofs of Theorem 12.3.1 and Theorem 12.4.1. By plugging in the result of Lemma 12.4.4, the claim follows. □

12.5 Scheduling to Minimize Weighted Modified Tardiness on Unrelated Machines

We now consider a generalization that aims to minimize weighted modified total tardiness on unrelated machines. Here, the processing speed of the jobs is different on different machines. For this, we rely on the algorithm and analysis by Megow et al. [9].

We consider the model where an algorithm can access a predicted processing speed for each machine and job. We represent this as $\hat{s_{ij}}$ while s_{ij} is the actual processing speed. Additionally, we assume a clairvoyant model, i.e., each job's exact (normalized) processing time is known upon arrival. Similar to the distortion parameter considered for the case of predicted job processing times, we let $\mu_1 = \max \frac{s_{ij}}{\hat{s_{ij}}}$ be the maximum underestimation error among all the arriving jobs. Similarly, we let $\mu_2 = \max \frac{\hat{s_{ij}}}{s_{ij}}$ be maximum overestimation error among all the speed predictions and $\mu = \mu_1 \cdot \mu_2$ be the distortion parameter.

In this section, we prove the following result.

Theorem 12.5.1. *Algorithm 4 is $O(\mu)$-competitive for the problem of minimizing weighted modified total tardiness on unrelated machines with speed predictions.*

12.5.1 Algorithm

The Maximum Density Scheduling Algorithm prioritizes jobs based on their density. The density is calculated for each active job by considering its weight, predicted speeds, and known processing times. The algorithm determines the maximum number of jobs to run based on available machines and the number of currently active jobs. It selects at most M jobs that maximize the sum of the densities to run.

12.5.2 Analysis

To prove Theorem 12.5.1, we need the following result.

Algorithm 4 Maximum Density Scheduling Algorithm

1: **function** UPONJOBRELEASE(j)
2: Compute density $\hat{\delta}_{ij} = \frac{w_j \hat{s}_{ij}}{p_j}$ for all active jobs
3: Find $k = \min(M, |J(t)|)$ jobs that maximize $(\sum_k \hat{\delta}_{ij})$
4: Run k jobs
5: **end function**
6: **function** UPONJOBCOMPLETION(j)
7: Compute density $\hat{\delta}_{ij} = \frac{w_j \hat{s}_{ij}}{p_j}$ for all active jobs
8: Find $k = \min(M, |J(t)|)$ jobs that maximize $(\sum_k \hat{\delta}_{ij})$
9: Run k jobs
10: **end function**

Lemma 12.5.2. *(Theorem 2.2 from [9]) Algorithm 4 has a competitive ratio of at most 8μ for minimizing the total weighted completion time on unrelated machines with speed predictions.*

Proof. See [9] for a comprehensive proof. □

Proof of Theorem 12.5.1 Let $C_j(alg)$ and $T_j(alg)$ be the completion times and tardiness of n jobs, respectively, due to the scheduling policy of Algorithm 1. Let $C_j(opt)$ and $T_j(opt)$ be the completion times and tardiness due to an optimal algorithm for the same instance of jobs.

By definition, we have the following equality.

$$\sum w_j(T_j(alg) + d_j) = \sum w_j \max(C_j(alg), d_j) \tag{12.22}$$

Similarly,

$$\sum w_j(T_j(opt) + d_j) = \sum w_j \max(C_j(opt), d_j) \tag{12.23}$$

The left-hand side of Eq. (12.22) represents the total weighted modified tardiness as the sum of weighted tardiness of individual jobs and their respective deadlines. This is equivalent to the weighted sum of the maximum between the completion time $C_j(alg)$ and the deadline d_j of each job j.

We have the following inequality by splitting the sum of the right-hand side into two separate sums.

$$\sum w_j \max(C_j(alg), d_j) \leq \sum w_j C_j(alg) + \sum w_j d_j \tag{12.24}$$

Similarly, we have the following lower bound.

$$\sum w_j \max(C_j(opt), d_j) \leq \sum w_j C_j(opt) + \sum w_j d_j \tag{12.25}$$

From the definition of competitive ratio, we have the following inequality bounding the total completion time.

$$\sum w_j \max(C_j(alg), d_j) \leq c \cdot \sum w_j C_j(opt) + \sum w_j d_j \tag{12.26}$$

Dividing the above equation with the respective lower bounds, we get

$$\frac{\sum w_j \max(C_j(alg), d_j)}{\sum w_j \max(C_j(opt), d_j)} \leq \frac{c \cdot \sum w_j C_j(opt) + \sum w_j d_j}{\sum w_j C_j(opt) + \sum w_j d_j} \tag{12.27}$$

Splitting the right-hand side of the above inequality into two separate terms, we get

$$\frac{c \cdot \sum w_j C_j(opt)}{\sum w_j C_j(opt) + \sum w_j d_j} \leq c \tag{12.28}$$

Similarly,

$$\frac{\sum w_j d_j}{\sum w_j C_j(opt) + \sum w_j d_j} \leq 1 \tag{12.29}$$

Combining Eq. (12.28) and Eq. (12.29) and rearranging the terms in equation Eq. (12.27), we get

$$\sum w_j \max(C_j(alg), d_j) \leq (c+1) \cdot \sum w_j \max(C_j(opt), d_j) \tag{12.30}$$

Using the result from Lemma 12.5.2, we can replace the constant c with asymptotic bound $O(\mu)$ of Algorithm 4. Ignoring the constant factors, we have the following inequality.

$$\sum w_j \max(C_j(alg), d_j) \leq O(\mu) \sum w_j \max(C_j(opt), d_j) \tag{12.31}$$

Using the definition of Eq. (12.22) and Eq. (12.23) and rewriting the above equation, we get

$$\sum w_j (T_j(alg) + d_j) \leq O(\mu) \sum w_j (T_j(opt) + d_j) \tag{12.32}$$

The claim follows. □

12.6 Conclusion

We considered the problem of minimizing modified total tardiness on parallel machines with different levels of information about job processing times and provided algorithms with competitive guarantees. One natural direction is to address the generalized version of the problem by including weights on identical machines. Another direction is to relax the requirement of clairvoyance for scheduling on unrelated machines.

12.7 Acknowledgement

The research leading to these results has received funding from the Knowledge Foundation (KKS), under the projects FIESTA (Project No. 20190034) and SACSys (Project No. 20190021), and under the Swedish Research Council (VR), under the project PSI (Project No. 2020-05094).

Bibliography

[1] Mikhail Y Kovalyov and Frank Werner. Approximation schemes for scheduling jobs with common due date on parallel machines to minimize total tardiness. *Journal of Heuristics*, 8:415–428, 2002.

[2] Stavros G Kolliopoulos and George Steiner. Approximation algorithms for scheduling problems with a modified total weighted tardiness objective. *Operations research letters*, 35(5):685–692, 2007.

[3] Ming Liu, Yinfeng Xu, Chengbin Chu, and Feifeng Zheng. Online scheduling to minimize modified total tardiness with an availability constraint. *Theoretical computer science*, 410(47-49):5039–5046, 2009.

[4] Yossi Azar, Eldad Peretz, and Noam Touitou. Distortion-Oblivious Algorithms for Scheduling on Multiple Machines. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 248, pages 16:1–16:18, 12 2022.

[5] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. *ACM Transactions on Parallel Computing*, 10(4):1–26, 2023.

[6] Nikhil Bansal. *Algorithms for flow time scheduling*. Carnegie Mellon University, 2003.

[7] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Semi-clairvoyant scheduling. *Theoretical computer science*, 324(2-3):325–335, 2004.

[8] Stefano Leonardi. A simpler proof of preemptive total flow time approximation on parallel machines. In *Efficient Approximation and Online Algorithms: Recent Progress on Classical Combinatorial Optimization Problems and New Applications*, pages 203–212. Springer, 2006.

[9] Alexander Lindermayr, Nicole Megow, and Martin Rapp. Speed-oblivious online scheduling: knowing (precise) speeds is not necessary. In *International Conference on Machine Learning*, pages 21312–21334. PMLR, 2023.