Accepted at: 4th IEEE International Conference on Edge Computing (EDGE 2020), pp. 1–9

Towards Extensibility-Aware Scheduling of Industrial Applications on Fog Nodes

Mohammadreza Barzegaran*, Vasileios Karagiannis†, Cosmin Avasalcai†
Paul Pop*, Stefan Schulte† and Schahram Dustdar†
*DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
{mohba, paupo}@dtu.dk
†Distributed Systems Group, TU Wien, Vienna, Austria
{v.karagiannis, c.avasalcai, s.schulte, dustdar}@dsg.tuwien.ac.at

***Abstract*—Fog computing has been identified as an enabler for many modern technologies like connected vehicles and the Industrial Internet of Things (IIoT). Such technologies are characterized by the integration of applications with different levels of criticality on shared platforms, which are referred to as mixed-criticality systems. Mixed-criticality systems typically use static scheduling for critical tasks; however, static scheduling is not suitable for scenarios where fog nodes run dynamic non-critical applications that implement, e.g., maintenance checks and data analytics.**

To address this challenge, in this paper, we differentiate between critical tasks that are statically allocated (called "native") and dynamic non-critical tasks that may migrate across fog nodes (called "temporary"). We propose a static scheduling approach that maximizes the number of temporary tasks that can be added at runtime, without negatively impacting the already scheduled native tasks. This approach enables fog nodes to become more suitable for IIoT environments by configuring them with extensible schedules for the native tasks. To evaluate our approach, we perform experiments considering several test cases, which show that given a number of native tasks, the generated extensible schedules enable the fog nodes to run a larger number of temporary tasks at the same time. Furthermore, the extensible schedules exhibit 7.8% less missed deadlines (on average), compared to the non-extensible schedules.

***Index Terms*—Fog computing, mixed-criticality systems, scheduling, extensibility, optimization**

I. Introduction

Computing at the edge of the network has emerged as a promising paradigm for enabling applications in various domains such as connected vehicles [1] and the Industrial Internet of Things (IIoT) [2]. According to this paradigm, the applications that demand guarantees in safety, security, and real-time performance, are executed on a *Fog Computing Platform* (*FCP*) which is extended from Cloud computing towards the edge of the network. Other terms (e.g. *Edge Computing*) with similar objectives and principles are also used for such platforms [3]. An *FCP* includes nodes capable of communicating and executing computations, i.e., *fog node*s (*FN*s), in the proximity of the data source [4] to guarantee effective collaboration between the devices, nodes and the Cloud (see Figure 1). An *FN* is a compute node that runs latency-sensitive applications on the available resources at the edge of the network [5]. *FN*s which intend to cope with low latency, need to consider that involved applications may have different criticality levels. Systems that host applications with different levels of criticality on the same platform, are usually referred to as *mixed-criticality systems* [6]. There are usually three levels of criticality in mixed-criticality systems: (i) safety-critical, (ii) mission-critical, and (iii) non-critical [6]; considering their timing requirements that may (i) compromise the safety of their surroundings, (ii) compromise their normal operation, and (iii) be prone to delay.

Even though coping with mixed-criticality applications is still a significant challenge in industrial Internet [7], both the Edge Computing consortium [8] and the Industrial Internet consortium [3] consider computing at the edge of the network as an enabler of smart factories which are at the very core of industrial Internet. So far, various methods have been proposed for scheduling tasks in mixed-criticality systems, such as hierarchical scheduling [9], task partitioning and scheduling [10], and container-based scheduling [11]. In addition, some approaches target *FN*s for mixed-criticality systems [12], and automotive use cases [13] in particular. However, none of these approaches consider the execution of dynamic fog applications that produce tasks which migrate across *FN*s. These applications are crucial for the operation of the IIoT since it enables the system to perform tasks such as maintenance checks, analytics, and software updates. For this reason, in this work, we design a scheduling algorithm for *FN*s, which considers both static critical tasks, i.e., the required tasks to execute industrial applications, which we call "native tasks"; and dynamic non-critical fog applications that implement, e.g., analytic services and diagnostic checks, which contain tasks that we call "temporary".

To achieve this, we bring *extensibility* [14], [15] to the schedules of *FN*s which supports adding a larger number of temporary tasks to the *FN* without disrupting the execution of the native tasks, i.e., without modifying the existing schedules. Keeping existing schedules unmodified is desirable since it preserves the performance level of the native tasks (including critical control applications) [16], and does not require re-certification of safety-critical applications. In particular, we address the problem of scheduling optimization of native tasks which are all critical with different levels of criticality on *FN*s, considering the extensibility of the schedules.

More precisely, our contributions are: We motivate the need for a novel scheduling optimization approach for *FN*s. We propose such a scheduling approach for optimizing extensibility of *FN*s' schedules, for which we have defined an extensibility metric. The proposed scheduling approach uses static cyclic scheduling policy to generate schedules which contain tasks' activation times. We also allow preemption in our schedules which is decided at the time of generating the schedules and increases the solution space. Besides, we propose an approach for scheduling of temporary tasks which considers the existing schedules and runs at runtime. Moreover, we provide an evaluation that compares the different derivatives of our proposed approach and shows the extensibility improvement.

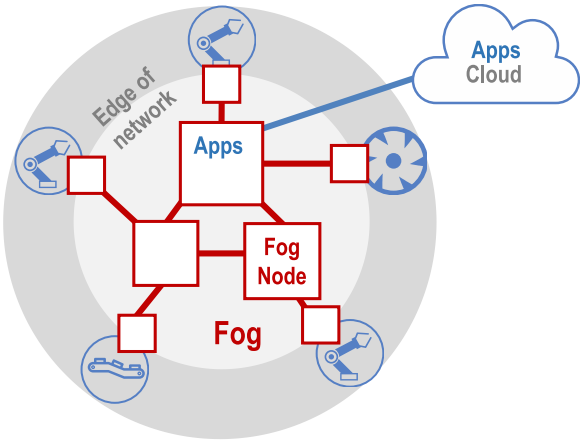


Fig. 1. Fog Computing Platform: *FN*s (boxes) are places at the edge of the network and connected via network (thick lines) to each other, equipment and the Cloud. Applications are running on *FN*s and in the Cloud.

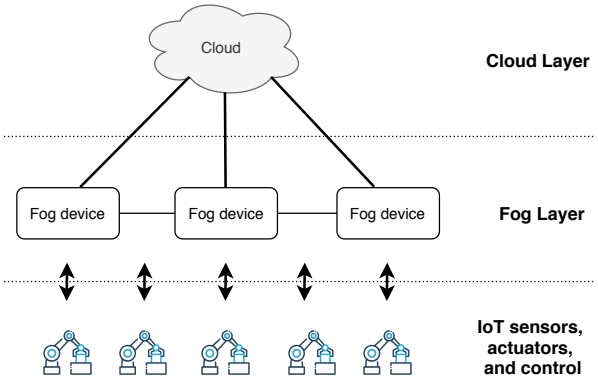


Fig. 2. Smart factory: an overview of the *FCP*.

The rest of the paper is structured as follows. Section II motivates our work by presenting the need for extensible schedules on *FN*s, and Section III describes the related work. Afterward, Section IV presents our underlying system model. We give the problem definition in Section V, and Section VI proposes our scheduling optimization approach. Finally, Section VII evaluates the performance of our proposed approach, and Section VIII concludes the paper.

II. Motivation

Current computing platform architectures in industry consist of different types of processing elements, e.g., programmable logic controllers or industrial personal computers, which run tasks that are typically high-critical [17]. Moreover, the processing elements currently used in industry have major limitations, such as lack of network capabilities, no support for web services, limited resource capacities, or plug-and-play features [17]. The paradigm of fog computing not only eliminates these limitations but also brings increased productivity and flexibility, mass customization, reduced time-to-market, improved product quality, innovations, and new business models [18], being an architectural means to realize the Operational and Information Technologies (OT/IT) convergence [19].

Both industry and academia have made a significant effort to promote the convergence of OT/IT. Until recently, OT was needed in small scale static networks within the industrial domain, which is why there is no support for dynamic changes and online reconfigurations [20]. This inability of OT now becomes a necessity with the increase in network size [20]. For this reason, the convergence of OT/IT aims at integrating mechanisms from the OT field into IT use cases to cope with such concerns [20]. The outcome of this convergence can increase the connectivity, interoperability, and scalability of industrial systems [21].

In this paper, we assume that the OT/IT convergence has taken place implementing the proposed platforms such as [22], and *FN*s now replace the processing elements within the industrial domain. An example of this assumption, implemented in industrial use cases, is the solution developed by Nebbiolo [23], where *FN*s are used in robotic applications. These *FN*s implement scheduling algorithms that schedule the execution of mixed-criticality tasks. Moreover, the *FN*s can communicate with each other for potentially sharing the load of the applications and also, they may communicate with the Cloud for exploiting available resources there [24]. Figure 2 presents an overview of the *FCP* implemented in the considered industrial use case.

For this environment, at design time, tasks which are all critical with different levels of criticality are scheduled on an *FN*. However, we make the key observation that due to the dynamic nature of fog applications the *FN* may be required to run additional tasks at runtime, which have not been considered for scheduling at design time, e.g., periodic maintenance tasks. Such tasks are non-critical but may have timing requirements (e.g., deadlines), and in addition they may be temporary, i.e., they may be replaced by other such tasks. Moreover, there may be new temporary tasks that are added to the *FCP* at a later time when the static scheduling of critical native tasks is already in place. As a result, the execution of these tasks may affect the timing requirements of the existing native tasks and hence compromise their performance and safety, e.g., if the native tasks are part of safety-critical applications.

This work's motivation stems from the need for industrial equipment implemented as *FN*s to run mixed-criticality native tasks alongside the temporary tasks for the industrial domain similar to [25]. We aim to bring extensibility to *FN*s' schedules in order to accommodate temporary tasks into the static schedules of native tasks, and thus efficiently use the available resources of *FN*s. The advantages of the *FN*s over legacy industrial processing elements become more evident when bringing this extensibility feature to the *FN*' schedules.

As a motivational example, let us consider an industrial application with several robots which uses an *FCP* as the architecture of the computing platform (see Figure 2). The native tasks are the control operation of robots, which are implemented on the *FN*s of the *FCP*; the static scheduling of these tasks is determined at design time. This static schedule is designed to ensure the correct functionality of the safety-critical tasks on the current platform. However, after a period of time, the system engineers define another set of tasks for data analytics; tasks that must run on the *FCP* without modifying the current schedule of native tasks. The engineers need to run these temporary tasks to record data from *FN*s to the Cloud for later analysis. The deployment strategy used to place the temporary tasks on the *FCP* can be done at runtime using a decentralized resource allocation technique [26]. Since the *FN*s have extensible schedules, the analytic tasks can be

added to run on the *FN*s without disturbing the performance of the control operation of robots.

III. Related Work

There has been much research in the field of scheduling algorithms for critical applications [27]. Thus, there are various approaches in the literature, each one showing benefits when applied to the targeted environment. The scheduling of critical applications with strict execution requirements has been studied in [28], and specifically for mixed-criticality systems in [29]. Besides, the scheduling of critical applications considering shared computation and communication resources has been studied in [30]. Nevertheless, none of these approaches takes into account the potential benefits of extensibility which we consider in our work.

Various approaches in the literature propose scheduling algorithms which target fog and edge computing environments [31]. Motivated by the challenge of scheduling tasks with execution deadlines in IoT systems, Deng et al. [32] describe a workload scheduling algorithm for delay-sensitive IoT applications in fog computing. In this work, the workload dynamic scheduling algorithm (WDSA) is designed to provide task scheduling toward worst-case delay and optimal utility for single hop Fog-IoT architectures. However, WDSA does not allow to schedule additional tasks without rescheduling the existing tasks. Barzegaran et al. [33] propose a scheduling algorithm for industrial applications in an *FCP* considering extra–functional requirements of the industrial control tasks. The proposed algorithm generates static cyclic schedules for the *FN*s, and also determines the mapping of tasks to the cores of *FN*s. In the work at hand, we use the same heuristic-based algorithm to generate static cyclic schedules. We consider the extensibility of the schedules, and ignore the problem of task mapping to cores. Moreover, we define a metric for extensibility of the schedules in Section VI-B which is used in our proposed algorithm as the optimization objective.

Yin et al. [11] propose a task scheduling algorithm for delay-sensitive applications in an *FCP*, which schedules the tasks either on the *FN*s or in the Cloud. This algorithm targets smart manufacturing environments that can benefit from the computational and storage services of fog computing, e.g., for fault detection or analysis of the state of the devices in assembly lines. This task scheduling algorithm is modeled considering the role of containers to cope with the limited resources and the delay-sensitive services that hinder the application of virtualization technologies in the scheduling. However, this work does not address scenarios where tasks may need to be scheduled on the *FN* temporarily, which is the goal of our work.

Pham and Huh [34] design a task scheduling algorithm for Cloud-fog computing environments. In this work, the authors first formulate a task scheduling problem, and then they propose a heuristic-based algorithm to balance execution time and cost. The proposed algorithm is intended to be used by a fog provider who wants to exploit both proprietary *FN*s and leased Cloud nodes in order to perform application offloading efficiently. Notably, the authors target scenarios such as a shopping center that has many *FN*s deployed on different floors for providing WiFi access and for delivering services. Thus, critical applications that have strict latency requirements are not considered.

Pop et al. [15] propose an incremental scheduling algorithm for embedded systems which aims at facilitating applications with deadlines. This approach considers a system with tasks that have already started and generates extensible schedules for adding specific future tasks considering that the existing tasks should be disturbed as little as possible. In this work, the authors use the idle time slots of the schedules in order to provide extensibility. For the evaluation, an extensibility metric is defined as the distribution of the idle time slot profiles in the schedules concerning the future task sets.

Various other metrics have also been defined for extensible schedules, such as the metrics in [14], [35]. Zhu et al. [35] propose an approach for robust task scheduling in distributed systems concerning the changes in task requirements. In this work, the notion of extensibility is used for robustness. The extensibility metric is defined as the weighted sum of each task's execution idle time over its period.

Zheng et al. [14] propose a mathematical modeling approach for extensible scheduling to accommodate additional tasks. In this work, the extensibility metric is defined as the maximum execution time a schedule can accommodate for a new independent task with certain period. This definition distributes the idle time among all tasks and targets all variations of future task sets, i.e., no prior specification of future tasks are required. In our work, we define extensibility in a way that our proposed approach provides general solutions independent of the additional tasks' specifications.

TABLE I
Summary of notation

Symbol	Notation
$\mathcal{N}$	Set of all fog nodes
$N_i \in \mathcal{N}$	fog node
$\mathcal{S}_{temp}$	Set of temporary schedule tables
$\mathcal{S}_{ext}$	Set of extensible schedule tables
$s_j \in \mathcal{S}_{temp}$	Temporary schedule table
$s_i \in \mathcal{S}_{ext}$	Extensible schedule table
$\upsilon_j \in s_i$	Execution slice
$\|s_i\|$	Number of execution slices
H	Hyperperiod
Γ_{native}	Set of all native applications
Γ_{temp}	Set of all temporary applications
γ_i	Set of tasks
$\tau_j \in \gamma_i$	Task
D_j	Deadline of a task
T_j	Period of a task
$\mathcal{C}_j$	WCET of a task
$\mathcal{M}$	The task mapping function to the fog nodes

IV. System Model

We assume an *FCP* with various *FN*s located at the edge of the network, and each *FN* integrates the proposed scheduling algorithm in order to handle critical tasks at design time, while non-critical tasks may also be added to each *FN* at runtime.

The system model consists of an architecture model that describes the architecture of the *FCP* and its *FN*s, and an application model which describes the tasks. Table. I summarizes the notation.

A. Architecture Model

The architecture consists of a set of *FN*s denoted by $\mathcal{N}$. Without loss of generality, we assume that each node $N_i \in \mathcal{N}$ has a single core CPU (multi-core *FN*s can be modeled by adding multiple nodes to the architecture). We use static cyclic scheduling to schedule the tasks once at design time and once at runtime, concerning the schedule tables.

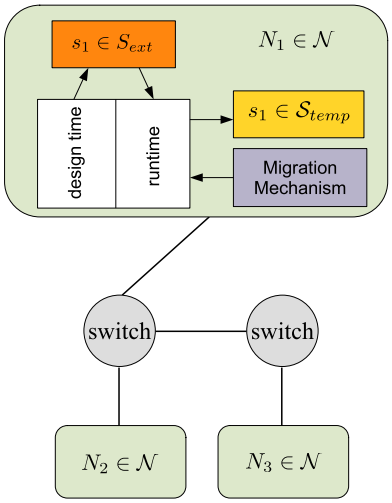


Fig. 3. Example architecture with three *FN*s: The orange box shows the extensible schedule table; the white box shows our proposed algorithm; the purple box shows the migration mechanism [26]; the yellow box shows the runtime schedule table.

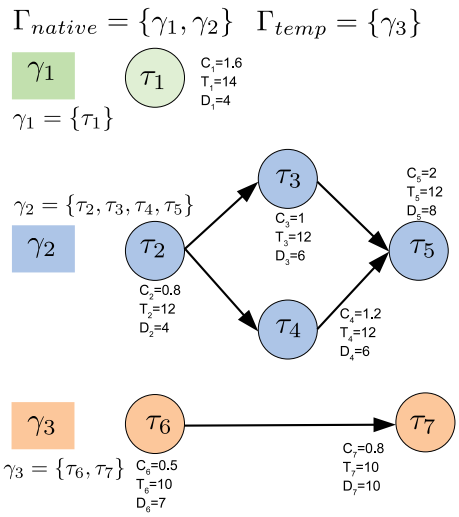


Fig. 5. Example application model with three applications and seven tasks

The set of all the schedule tables which will be determined by our proposed scheduling approach for extensibility at design time, is denoted with $\mathcal{S}_{ext}$. We call these schedule tables "extensible". Our proposed scheduling approach may be run at runtime to schedule the temporary tasks (see Section IV-B) which determines the set of the schedule tables, denoted with $\mathcal{S}_{temp}$. We call these schedule tables "temporary".

A static schedule table repeats with a hyperperiod denoted by H, which is the least common multiple of all the assigned task periods. To increase the design space of our solution, we allow tasks to preempt each other in the generated schedule, and thus our proposed approach may decide to execute a task in several execution slices. The preemption granularity is controlled by a parameter called macrotick [36], thus improving the schedulability [37] and performance of control tasks [16].

An example architecture model composed of three *FN*s is shown in Fig. 3. The extensible schedule table (orange box) is determined by our proposed algorithm (white box) during design time. At runtime, our proposed algorithm schedules the temporary tasks which have arrived with a migration mechanism (purple box), and determines the temporary schedule table (yellow box).

The schedule table s_i has a set of execution slices denoted by $\upsilon_j \in s_i$, which are time slices representing the task executions generated by capturing the start time and finishing time of the tasks assigned to the core of N_i. We denote the number of execution slices in the schedule table with $|s_i|$. We give an example extensible schedule table $s_1 \in \mathcal{S}_{ext}$ of the node N_1, in Fig. 4 using a Gantt chart, where the boxes are execution slices, and the arrows show task preemption. The example has three tasks denoted with different colors with a total of

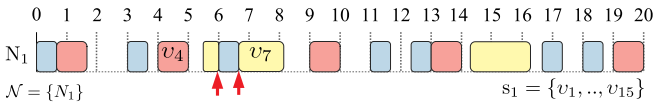


Fig. 4. Example schedule table of N_1 with three tasks and fifteen execution slices: different colors represent different tasks; boxes show execution slices; arrows show preemption.

fifteen execution slices. The blue task interrupts the yellow task as indicated by arrows. An extensible schedule table has distributed idle times. Thus, an additional task can use the idle times to be executed before its deadline (see extensibility metric in Section. VI-B).

B. Application Model

Our application model consists of (i) a set of applications considered at design time, denoted with Γ_{native}, and (ii) a set of applications considered at runtime, denoted with Γ_{temp}. Each native application $\gamma_i \in \Gamma_{native}$ is statically allocated to an *FN* at design time, and consists only of native tasks which are static critical tasks with different levels of criticality. Furthermore, each temporary application $\gamma_j \in \Gamma_{temp}$ will be dynamically allocated to an *FN* and may migrate across the *FN*s. The application consists only of temporary tasks which are dynamic non-critical tasks.

Each application γ_i (whether native or temporary) is modeled with a directed acyclic graph (DAG), where nodes and edges represent tasks and data flows between the tasks. A task $\tau_j \in \gamma_i$ has a period T_j, a deadline D_j, and a known worst-case execution time (WCET) C_j on the mapped *FN*. Each task is ready to execute when all its inputs have arrived. The output of a task is produced upon the termination of the task. The mapping of tasks to the *FN*s is known and captured by the function $\mathcal{M} : \tau_i \longrightarrow \mathcal{N}$.

An example application model composed of three applications (and seven tasks) is shown in Fig. 5. The application γ_3 and its tasks are temporary and the remainders are native. The WCETs, periods, and deadlines of the tasks in milliseconds are shown in the figure.

V. Problem Definition

We formally define the problem to be solved by our approach as follows. Given (i) a set of native applications Γ_{native}, (ii) a set of temporary applications Γ_{temp}, (iii) an architecture consisting of a set of fog nodes $\mathcal{N}$, and (iv) the mapping of tasks to the *FN*s captured by the function $\mathcal{M}$, we want to determine: (i) The set of extensible schedule tables $\mathcal{S}_{ext}$ at design time. (ii) The set of temporary schedule tables $\mathcal{S}_{temp}$ at runtime. These need to be determined such that: (1) the deadlines of all the tasks (both native and temporary) are met, and (2) the extensibility of the extensible schedules is maximized,

Algorithm 1 $\mathcal{S} = EASA(\Gamma, \mathcal{N}, \mathcal{M}, \Psi)$

1: $i \leftarrow 0$
2: $t \leftarrow T_{start}$
3: $\Phi \leftarrow \{D_i\}; \Theta \leftarrow \{0\}$
4: $\mathcal{S} \leftarrow$ EDFSimulation$(\Gamma, \mathcal{N}, \mathcal{M}, \Phi, \Theta, \Psi)$
5: $\Omega \leftarrow$ CostFunction$(\mathcal{S}, \Gamma)$
6: **repeat**
7: $< \Phi_i, \Theta_i > \leftarrow$ Neighbor$(\Gamma, \mathcal{N}, \mathcal{M}, \Phi, \Theta)$
8: $\mathcal{S}_i \leftarrow$ EDFSimulation$(\Gamma, \mathcal{N}, \mathcal{M}, \Phi_i, \Theta_i, \Psi)$
9: $\Omega_i \leftarrow$ CostFunction$(\mathcal{S}_i, \Gamma)$
10: $\lambda \leftarrow \Omega_i - \Omega$
11: **if** $\lambda < 0$ **or** $random[0,1) < Prob(\lambda, t)$ **then**
12: $\mathcal{S} \leftarrow \mathcal{S}_i; \Phi \leftarrow \Phi_i; \Theta \leftarrow \Theta_i$
13: **end if**
14: $t \leftarrow t \times \alpha$
15: **until** stopping criterion is true
16: **return** $\mathcal{S}$

which enables adding a larger number of temporary tasks at runtime.

VI. Proposed Solution

In this section, we first describe our scheduling optimization approach in detail and then define our objective function, which considers extensibility with an example to support.

A. Scheduling algorithm

The proposed approach, named Extensibility–Aware Scheduling Algorithm (*EASA*), is a Simulated Annealing (*SA*)-based metaheuristic [38]. *SA* is an optimization heuristic that tries to find the global optimum by randomly selecting a new solution from the neighbours of the current solution [38]. *SA* uses *moves* to explore the search space and to generate solutions from the neighbours which are evaluated with the objective function, presented in Section VI-B. Naturally, *SA*, as a heuristic, is not guaranteed to find the optimal solution.

The *SA* decides scheduling parameters which are task offsets (earliest start time of tasks) and relative deadlines, denoted by Θ and Φ respectively. The schedule tables are generated based on an *EDF* simulation [27], using the scheduling parameters which indicate when tasks become ready (based on their offsets) and the priority of the ready tasks (based on their relative deadline) for running at the time being. Deciding different scheduling parameters generates different schedules, thereby *SA* explores the search space to try finding the global optimum. Our proposed *EASA* (shown in Algorithm 1) has two variants which can be applied: (i) at design time for determining the extensible schedules, and (ii) at runtime for determining the temporary schedules. The configuration for design time variant is: (1) $\Gamma \longleftarrow \Gamma_{native}$, (2) $\mathcal{S}_{ext} \longleftarrow \mathcal{S}$, and (3) $\Psi \longleftarrow \varnothing$. The configuration for the runtime variant is accordingly: (1) $\Gamma \longleftarrow \Gamma_{temp}$, (2) $\mathcal{S}_{temp} \longleftarrow \mathcal{S}$, and (3) $\Psi \longleftarrow \mathcal{S}_{ext}$.

The *EASA* starts from a solution with the initial scheduling parameters: the offsets Θ and relative deadline Φ are set to zero and their deadline values, respectively (line 3 in Alg. 1), and explores the solution space (lines 6–15). The schedule tables $\mathcal{S}$ are generated from the *EDF* simulation (line 4) which is performed for a hyperperiod H (see Section IV-A).

The *EDF* simulation creates jobs of tasks on the fly, and gives the highest priority to the job which has the earliest deadline with all its precedent jobs having arrived at the time being. High-priority jobs run on the mapped cores captured by the function $\mathcal{M}$ (for the design time variant: at the time being; for the run time variant: at the earliest time when the associated core becomes idle concerning the extensible schedules) for the duration of their WCETs C_i. A high-priority ready job may interrupt the currently running job (on the same core) if its priority is higher considering the scheduling constraints, e.g., the data dependency (see Section IV-B) and the macrotick (see Section IV-A). In the runtime variant of *EASA*, an idle space coming to the end interrupts the currently running job (on the same core), forcing the job to continue its execution from the next idle space concerning the extensible schedules.

A new solution from the neighbours of the current solution is generated using the moves iteratively which randomly varies the scheduling parameters (line 7). The *Deadline Adjustment* move randomly selects a task and sets its relative deadline Φ in the range from its deadline D_i to its period T_i. The *Offset Adjustment* move randomly selects a task and sets its offset Θ in the interval from 0 to its deadline D_i.

The new solution is evaluated with the objective function Ω (line 9), see Section VI-B. *EASA* compares the Ω_i value of the new solution with the Ω value of the current solution, and accepts the new solution if the cost is improved (line 11). A new solution may be accepted with a certain probability even if the cost is not improved, thus better exploring the solution space. The acceptance probability is

$$Prob(\lambda, t) = e^{-\frac{\lambda}{t}}, \tag{1}$$

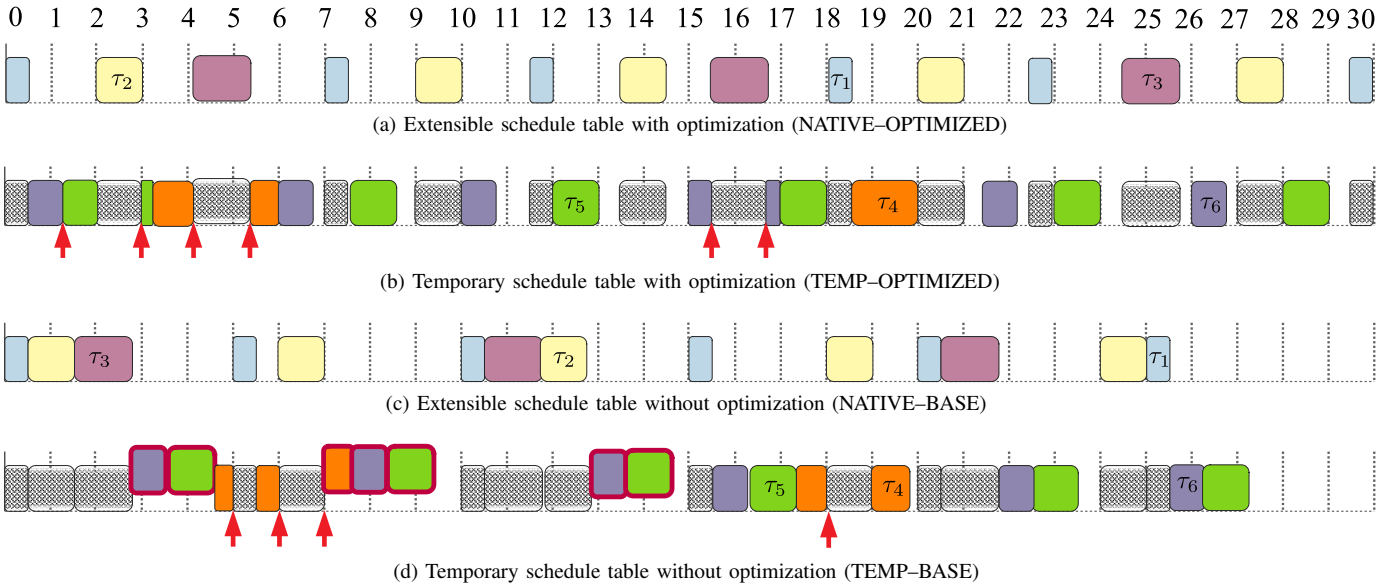
where λ is the cost difference calculated in line 10. It decreases as the temperature t cools down from an initial temperature T_{start} (line 2) with the rate of α as the time passes in each iteration (line 14). A stopping criterion stops the search (line 15). In the work at hand, the stopping criterion is either that no improvement after a given number of iterations occurs, that a temperature of zero is reached, and a time limit. The one happening first is applied as stopping criterion.

B. Extensibility metric and objective function

We define the objective function Ω for evaluating the solutions generated by *EASA*, in Eq.(2). The objective function takes the schedule tables ($\mathcal{S}_{ext}$ in the design time variant and $\mathcal{S}_{temp}$ in the runtime variant) and calculates the cost with the weighted summation of two terms: the extensibility metric and task schedulablity constraints. The *EASA* controls the search for schedulable solutions with optimized objective decided with weights by choosing a lager β_1 to the search for optimized extensibility, and contrariwise for finding schedulable solutions faster. The weight β_1 is always zero in the runtime variant of *EASA*.

$$\Omega = \beta_1 \times E + \beta_2 \times \Lambda \tag{2}$$

Our proposed extensibility definition is similar to the one in [14]: The extensibility is captured by the function E which is shown in Eq. 3. The function E first finds the idle time slices in each schedule table $s_i \in \mathcal{S}_{ext}$, by removing the execution slices $\upsilon_j \in s_i$ from it (equivalent to the time slices in which the assigned core is idle), denoted with $\mathcal{L}_k^i$. These time slices – which are "idle spaces" – are where the temporary tasks (assigned to the same *FN*) can run. The function E calculates the variance of the duration of the idle spaces. A solution with better extensibility has less deviation in the duration of all idle spaces in each schedule table, thus, a smaller value of function E.



(a) Extensible schedule table with optimization (NATIVE–OPTIMIZED)

(b) Temporary schedule table with optimization (TEMP–OPTIMIZED)

(c) Extensible schedule table without optimization (NATIVE–BASE)

(d) Temporary schedule table without optimization (TEMP–BASE)

Fig. 6. Four schedule tables for an illustrative example: NATIVE–OPTIMIZED is optimized for extensibility, TEMP–OPTIMIZED has successfully added temporary tasks (hatched boxes represent native tasks), NATIVE–BASE does not consider extensibility, TEMP–BASE cannot successfully add temporary tasks, thus, some tasks (shown with red border boxes) have missed their deadlines (hatched boxes represent native tasks).

$$
\begin{aligned}
&\forall s_i \in \mathcal{S}_{ext}, m = |s_i| - 1, j = \{1, .., m\}, \upsilon_j \in s_i, \\
&\mathcal{L}_k^i = \left| start_{\upsilon_{j+1}} - end_{\upsilon_j} \right|, \\
&\bar{\mathcal{L}} = \frac{\sum \mathcal{L}_k^i}{m}, \sigma_{\mathcal{L}} = \sqrt{\frac{\sum |\mathcal{L}_k^i - \bar{\mathcal{L}}|}{m}} : \\
&E = \sigma_{\mathcal{L}} \times H^{-1}
\end{aligned}
\tag{3}
$$

We present an illustrative example for the extensibility metric in Fig. 6 where the schedules of the tasks from Table II are shown. The colored boxes represent different tasks. We generate an optimized extensible schedule for native tasks (described in Fig. 6a as "NATIVE-OPTIMIZED"), and a non-optimized extensible schedule table (described in Fig. 6c as "NATIVE-BASE"). We applied the function E to both "NATIVE-OPTIMIZED" and "NATIVE-BASE", which have thirteen and eight number of idle spaces respectively. All native tasks are successfully scheduled, and none of them misses its deadline in both schedules. The function E reports the values of 0.0035 and 0.0420 for "NATIVE-OPTIMIZED" and "NATIVE-BASE" respectively, which shows that the duration of idle spaces in "NATIVE-OPTIMIZED" is less deviated.

We generate temporary schedule tables at runtime for the temporary tasks based on their existing schedules. We show the temporary schedule table "TEMP-OPTIMIZED" in Fig. 6b and the non-optimized version "TEMP-BASE" in Fig. 6d. All temporary tasks are successfully scheduled, and none of them misses its deadline in "TEMP-OPTIMIZED", whereas, the scheduling is not successful in "TEMP-BASE" and three tasks miss their deadlines.

The task schedulablity constraints are captured by the function Λ, which checks for deadline violations of all the tasks in the schedule tables. Since the outcome of the function E is in the range $[0,1]$, we normalize Λ starting form 0, for no violations, to 1, for the case in which all the jobs have missed their deadlines.

TABLE II
DETAILS OF THE ILLUSTRATIVE EXAMPLE

Application set	Applications	Tasks	C (μs)	T (μs)	D (μs)
Γ_{native}	γ_1	τ_1	500	5000	4000
		τ_2	1000	6000	4000
		τ_3	1200	10000	9000
Γ_{temp}	γ_2	τ_4	1500	15000	7000
		τ_5	1000	5000	4000
		τ_6	750	5000	3000

VII. Evaluation

Our proposed scheduling optimization approach, *EASA*, was implemented in C# and all the experiments were run on a computer with an i9 CPU at 3.6 GHz and 32 GB of RAM, with a time limit of 10 to 30 minutes, depending on the size of the test case.

The weights of the objective function Ω were determined experimentally to guide the search faster towards feasible solutions with optimized extensibility. We set the weight β_1 and β_2 to 0.25 and 1.0, respectively. The weight β_1 can be determined by analyzing periods and WCETs of native and temporary tasks. The value of 1.0 for the weight β_2 is a relatively large value considering that the terms of the objective function are in the range $[0,1]$.

We have evaluated our proposed method on eight synthetic test cases and one realistic test case. We have generated the synthetic test cases considering the overall CPU utilization of all their tasks and progressively increasing number of native tasks and applications. The details of the synthetic test cases are shown in Table III. The realistic test case is an *FN* inside a vehicle [39] which has three native applications (representing different engine control applications) and two temporary applications (representing different passenger comfort applications). The details of the realistic test case are shown in Table IV.

A. Synthetic Test Cases

In this first set of experiments we were interested to evaluate the ability of our proposed method to create extensible schedules of native tasks that accommodate adding a larger number of temporary tasks. The focus is on evaluating our algorithm,

TABLE III
EVALUATION RESULTS FOR THE EIGHT SYNTHETIC TEST CASES

Test cases	Total no. of native applications/tasks	Total no. of temporary applications/tasks	CPU Utilization	Ω of EASA	Perc. dev. Ω of EASA/E from EASA	Missed deadlines with EASA	Missed deadlines with EASA/E	Missed deadlines with EASA/R
1	3/7	2/3	63%	0.0680	263%	0%	0%	0%
2	3/7	4/8	79%	0.0680	263%	0%	0%	0%
3	3/8	4/8	80%	0.0580	324%	0%	38%	0%
4	3/8	2/6	69%	0.0580	324%	0%	0%	0%
5	4/10	2/6	82%	0.0420	307%	0%	0%	0%
6	4/10	3/4	83%	0.0420	307%	0%	8%	0%
7	5/12	3/4	94%	0.0397	442%	4%	16%	0%
8	5/13	2/3	90%	0.0417	436%	0%	4%	0%
Average			80%	0.0522	333%	0.5%	8.3%	0%

hence look in isolation at a single fog node and consider that the created schedule has to accommodate all temporary applications from Table III on this fog node throughout the execution. In order to facilitate evaluation, we have defined two other solutions for comparison purposes: *EASA/E*, and *EASA/R*.

The *EASA/E* is derived from *EASA* by ignoring the extensibility optimization. In this solution, the schedules are generated concerning only the schedulability constraints by setting the weight β_1 to zero which removes the extensibility metric value E in the evaluation of the visited solution. Hence, leaving the weight β_2 as the same value for *EASA*. Such a solution could be in principle generated by a system engineer with the state-of-the-art scheduling tools on the market.

The *EASA/R* is also derived from *EASA* by unifying both temporary and native tasks and ignoring the extensible schedules. The solution's schedules are generated at runtime by rescheduling both native and temporary tasks. Hence the weight β_1 is set to zero (ignoring the incremental scheduling aspect) and the value of the weight β_2 is the same as *EASA*'s. The idea behind this solution is to adapt at runtime to changes in tasks, i.e., instead of focusing of creating extensible schedules at design time the alternative implemented by *EASA/R* is to allow the schedules to completely change at runtime, including allowing changes to native tasks.

The evaluation results are also presented in Table III. We report the objective function values Ω for the extensible schedules of *EASA* in column 5. The column 6 reports the variation of objective function value for *EASA/E* comparing to the values of *EASA*. No values are reported for *EASA/R*, since the extensibility is not applicable to the solutions.

Since all the three solutions are considering the scheduling constraints, we have reported the percentage of missed scheduling constraints (missed task deadlines) for all the tasks (both native and temporary tasks) in the table. In all test cases the native tasks always meet their deadlines (there is enough capacity for the native tasks in the fog node since it is expected to handle additional temporary applications), so the percentage of missed deadlines shows the deadlines missed by temporary tasks.

As the results show, no missed deadlines are reported for *EASA/R* which means that all the generated schedules are feasible. This may indicate that it is preferable to re-generate all schedules at runtime every time there is a change. However, for the safety-critical native applications the safety standards dictate the development processes and the certification procedures that have to be followed, increasing the cost of the system. Furthermore, safety assurance practice dictates that changes to a certified system result in the re-certification of the system, adding further (re-certification) costs. For this reason, changing the existing schedules of the native tasks at runtime is not desirable, i.e., they have to be fixed at design time. Also, by creating schedules for all tasks in the system (both native and temporary) every time there is a change in temporary tasks allocated to the fog node will generate a larger computational overhead. Furthermore, when native tasks are scheduled at the same time with temporary tasks, we increase the risk of missing the deadlines for the native (critical) tasks, which renders them unsafe. It is preferable that we miss deadlines for the temporary tasks, which can be migrated to fog nodes with more resources in order to successfully execute.

EASA/E, which does not consider extensibility, results in missed deadlines for temporary tasks: up to 38% of deadlines are missed in test case 3, and 8.3% on average. This is expected, since *EASA/E* does not consider the need to extend the native tasks' schedules to accommodate temporary tasks. The important result here is that our proposed *EASA* solution that considers extensibility is able to successfully schedule both the native and temporary tasks in all test cases, with a single exception: 4% of deadlines are missed in test case 7 (the assumption is that such temporary tasks that miss deadlines could be migrated to another fog node with more resources). On average, *EASA* improves over *EASA/E* in terms of satisfying the number of task deadlines with 7.8%.

In the table we also show the value of the cost function Ω for *EASA* and the percentage deviation from this value of *EASA/E*. These Ω columns indicate the reason why we have less missed deadlines with *EASA* compared to *EASA/E*. *EASA* has been able to obtain an average objective function Ω of 0.0522, which shows the solutions have less deviated idle space duration, compared to the average value of 0.2261 for *EASA/E*. The Ω values for *EASA* are smaller in all the test cases and the *EASA*'s schedules are optimized for extensibility, thus allow adding more temporary tasks and less missed deadlines as a result.

B. Realistic Test Case: Fog-based vehicle applications

In the second set of experiments we were interested to evaluate our proposed solution on a realistic test case consisting of applications running on a fog node inside a vehicle. Future vehicles are envisioned to be "fog nodes on wheels" [39] as they integrate more and more functions and become interconnected with each other. In this test case we consider the dynamic nature of the fog, i.e., we evaluate our solution on scenarios where temporary applications migrate in-and-out of the fog node over time.

TABLE IV
FOG-BASED VEHICLE APPLICATIONS: SENSOR READING (SR); TEMPERATURE (TEMP); APPLICATION (APP.)

App set	Apps	Tasks	C (ms)	T (ms)	D (ms)
Engine control apps (Γ_{native})	Coolant temp. (γ_1)	Temp SR (τ_1)	0.7	20	18
		Fan speed set (τ_2)	0.4	20	20
	Coolant flow (γ_2)	Flow SR (τ_3)	0.3	16	14.4
		Throttle SR (τ_4)	0.6	16	15
		Valve relay set (τ_5)	1	16	16
	Oil flow (γ_3)	Oil flow SR (τ_6)	1.8	12	7
		Indicator set (τ_7)	1.5	12	11
Passenger comfort apps (Γ_{temp})	Climate control (γ_4)	Temp SR (τ_8)	0.1	30	18
		Input reading (τ_9)	0.1	30	19
		A/C set (τ_{10})	0.6	30	20
	Cabin light (γ_5)	Lighting SR (τ_{11})	0.2	30	23
		Lights set (τ_{12})	0.3	30	25

The details of the test case are in Table IV. The seven native tasks are distributed in three applications γ_1, γ_2, and γ_3. The *FN* on the vehicle is needed to run five temporary tasks in two application γ_4, γ_5 which may also be removed/replaced from the *FN* at runtime.

We scheduled the native tasks with our proposed method at design time, and generated the extensible schedules $\mathcal{S}_{ext}$. The generated solution successfully scheduled all the native tasks, i.e., no deadlines are missed. *EASA* reported a value of 0.0738 for the objective function Ω. We assume that the *FN* has to run the temporary application γ_4 at the passenger's request. *EASA* also successfully scheduled these temporary tasks at runtime generating the temporary schedules $\mathcal{S}_{temp}$. In our considered scenario, the temporary application γ_4 stops after some time. Thus, *EASA* removes its tasks and rolls back to the extensible schedules $\mathcal{S}_{ext}$. Afterwards, the application γ_5 is migrated to run on the *FN*, and *EASA* successfully extended the schedules $\mathcal{S}_{ext}$ to accommodate the tasks of γ_5. We also evaluated our proposed solution on the scenario where both applications γ_4 and γ_5 arrive at the same time. *EASA* also successfully scheduled all the temporary tasks.

As indicated by this experiment, our proposed *EASA* shows promising results in successful adding of temporary tasks without impacting the native tasks also in the case of realistic test cases. Besides, *EASA* being more successful in scheduling of temporary tasks, avoids negatively impacting the native tasks. The experiments in this section show that our proposed *EASA* method is able to bring extensibility to *FN*s' schedules.

VIII. CONCLUSIONS

Fog Computing is used in an increasing number of application areas, including in areas with safety- and time-critical applications. In the Industrial IoT area the vision is to virtualize industrial equipment and services, including critical control, and run these as software tasks on a Fog Computing Platform (FCP). However, in such application areas the resources of the FCP have to be statically allocated at design time to these critical in order to provide the dependability guarantees required by safety assurance. These applications then become native to (associated to) the fog nodes where they are allocated.

The disadvantage of the design time static configuration of the FCP is that it will not easily accommodate dynamic non-critical fog applications. Hence, in this paper, we have proposed a scheduling approach for fog nodes in an FCP that aims at increasing the extensibility of the static configuration. This approach can schedule statically allocated critical tasks in a way that allows adding more dynamic non-critical tasks without compromising the performance of the native tasks. This makes our proposed approach applicable to industrial environments where the fog nodes need to run critical tasks but also, they need to run non-critical tasks which are not considered at design time such as maintenance checks, analytics, etc. Our proposed method does not consider any assumption about temporary tasks and generates extensible solutions. To evaluate our approach, we evaluated its performance on several test cases. The results show the validity of our approach and its ability to synthesize extensible configurations, where more temporary tasks can be added at runtime.

In our future work, we will consider the temporal separation of tasks with different criticality levels in the extensible schedules. We will take into account the possibility of having isolated partitions for native and temporary tasks, and we will also consider using Constraint Programming as an alternative optimization technique, integrating both the scheduling of tasks and the scheduling of messages.

ACKNOWLEDGEMENTS

The research leading to these results 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 Industrial Automation.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Workshop on Mobile Cloud Computing (MCC)*, pp. 13–16, ACM, 2012.

[2] C.-H. Chen, M.-Y. Lin, and C.-C. Liu, "Edge computing gateway of the Industrial Internet of Things using multiple collaborative micro controllers," *IEEE Network*, vol. 32, no. 1, pp. 24–32, 2018.

[3] OpenFog Consortium, "OpenFog reference architecture for fog computing," *Technical Report*, 2017.

[4] V. Karagiannis, S. Schulte, J. Leitao, and N. Preguica, "Enabling fog computing using self-organizing compute nodes," in *International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, 2019.

[5] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[6] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 82, 2018.

[7] J.-Q. Li, F. R. Yu, G. Deng, C. Luo, Z. Ming, and Q. Yan, "Industrial Internet: A survey on the enabling technologies, applications, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1504–1526, 2017.

[8] Edge Computing Consortium, "Edge computing reference architecture 2.0," *Technical Report*, 2017.

[9] Y. Wang, Y. Zhang, Y. Su, X. Wang, X. Chen, W. Ji, and F. Shi, "An adaptive and hierarchical task scheduling scheme for multi-core clusters," *Parallel Computing*, vol. 40, no. 10, pp. 611 – 627, 2014.

[10] K. Lakshmanan, R. Rajkumar, and J. Lehoczky, "Partitioned fixed-priority preemptive scheduling for multi-core processors," in *Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 239–248, IEEE, 2009.

[11] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4712–4721, 2018.

[12] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog computing for the internet of things: A survey," *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–41, 2019.

[13] F. Bonomi, S. Poledna, and W. Steiner, *The role of fog computing in the future of the automobile*, pp. 189–210. Wiley Online Library, 2017.

[14] Wei Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Extensible and scalable time triggered scheduling," in *International Conference on Application of Concurrency to System Design (ACSD)*, pp. 132–141, 2005.

[15] P. Pop, P. Eles, and Z. Peng, "Incremental mapping and scheduling for distributed heterogeneous real-time systems," *Real-Time in Sweden*, 2003.

[16] M. Barzegaran, A. Cervin, and P. Pop, "Performance optimization of control applications on fog computing platforms using scheduling and isolation," *IEEE Access*, vol. 8, pp. 104085–104098, 2020.

[17] M. Jbair, B. Ahmad, M. H. Ahmad, and R. Harrison, "Industrial cyber physical systems: A survey for control-engineering tools," in *IEEE Industrial Cyber-Physical Systems*, pp. 270–276, 2018.

[18] H. Bauer, C. Baur, D. Mohr, A. Tschiesner, T. Weskamp, K. Alicke, and D. Wee, "Industry 4.0 after the initial hype–where manufacturers are finding value and how they can best capture it," *McKinsey Digital*, 2016.

[19] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling fog Computing for industrial automation through time-sensitive networking (TSN)," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 55–61, 2018.

[20] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of IEEE 802.1 TSN networks," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2017.

[21] O. Givehchi, K. Landsdorf, P. Simoens, and A. W. Colombo, "Interoperability for industrial cyber-physical systems: An approach for legacy systems," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3370–3378, 2017.

[22] P. Pop, B. Zarrin, M. Barzegaran, S. Schulte, S. Punnekkat, J. Ruh, and W. Steiner, "The FORA Fog Computing Platform for Industrial IoT," *arXiv preprint arXiv:2007.02696*, 2020.

[23] Nebbiolo Technologies, "Nebbiolo." http://www.nebbiolo.tech, 2020 (accessed May 10, 2020).

[24] V. Karagiannis, "Compute node communication in the fog: Survey and research challenges," in *Workshop on Fog Computing and the IoT (IoT-Fog)*, pp. 1–5, ACM, 2019.

[25] M. Barzegaran, N. Desai, J. Qian, K. Tange, B. Zarrin, P. Pop, and J. Kuusela, "Fogification of electric drives: An industrial use case," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, p. 1–5, 2020.

[26] C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Decentralized resource auctioning for latency-sensitive edge computing," in *IEEE International Conference on Edge Computing (EDGE)*, 2019.

[27] G. C. Buttazzo, *Hard real-time computing systems: Predictable scheduling algorithms and applications*. Springer, 2011.

[28] P. Naghshtabrizi and J. P. Hespanha, "Analysis of distributed control systems with shared communication and computation resources," in *American Control Conference (ACC)*, pp. 3384–3389, 2009.

[29] D. Tamas-Selicean and P. Pop, "Design optimization of mixed-criticality real-time systems," *ACM Transaction on Embedded Computing*, vol. 14, pp. 50–78, May 2015.

[30] D. Fontanelli and L. Palopoli, "Quality of service and quality of control in real-time control systems," in *International Symposium on Communications, Control and Signal Processing (ISCCSP)*, pp. 1–5, IEEE, 2012.

[31] J. Bellendorf and Z. Á. Mann, "Classification of optimization problems in fog computing," *Future Generation Computer Systems*, vol. 107, pp. 158–176, 2020.

[32] Y. Deng, Z. Chen, D. Zhang, and M. Zhao, "Workload scheduling toward worst-case delay and optimal utility for single-hop Fog-IoT architecture," *IET Communications*, vol. 12, no. 17, pp. 2164–2173, 2018.

[33] M. Barzegaran, A. Cervin, and P. Pop, "Towards quality-of-control-aware scheduling of industrial applications on fog computing platforms," in *Workshop on Fog Computing and the IoT (IoT-Fog)*, p. 1–5, ACM, 2019.

[34] X.-Q. Pham and E.-N. Huh, "Towards task scheduling in a Cloud-Fog computing system," in *Asia-Pacific network operations and management symposium (APNOMS)*, pp. 1–4, 2016.

[35] Q. Zhu, Y. Yang, E. Scholte, M. D. Natale, and A. Sangiovanni-Vincentelli, "Optimizing extensibility in hard real-time distributed systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 275–284, 2009.

[36] S. S. Craciunas, R. Serna Oliver, and V. Ecker, "Optimal static scheduling of real-time tasks on distributed time-triggered networked systems," in *International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, IEEE, 2014.

[37] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium (RTSS)*, pp. 239–243, IEEE, 2007.

[38] E. K. Burke, G. Kendall, *et al.*, *Search methodologies*. Springer, 2005.

[39] M. Chiang, B. Balasubramanian, and F. Bonomi, *Fog for 5G and IoT*, vol. 288. Wiley Online Library, 2017.