Journal of Systems Architecture 133 (2022) 102776

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Extensibility-aware Fog Computing Platform configuration for mixed-criticality applications

Mohammadreza Barzegaran *, Paul Pop

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

ARTICLE INFO

Keywords:
Fog Computing
Mixed-criticality systems
Scheduling
Extensibility
Evolvability
Optimization
Time-Sensitive Networking

ABSTRACT

In this paper, we consider that critical control applications and Fog applications share a Fog Computing Platform (FCP). Critical control applications are implemented as periodic hard real-time tasks and messages and have stringent timing and safety requirements, and require safety certification. Fog applications are implemented as aperiodic tasks and messages and are not critical. Such applications need different approaches to guarantee their timing and dependability requirements. We formulate an optimization problem for the joint configuration of critical control and Fog applications, such that (i) the deadlines and Quality-of-Control (QoC) of control applications are guaranteed at design-time, (ii) the configuration is extensible and supports the addition of future new control applications without requiring costly re-certification, and (iii) the design-time configuration together with the runtime Fog resource management mechanisms, can successfully accommodate multiple dynamic responsive Fog applications. We evaluate our approach on several test cases assuming scenarios for hosting both Fog applications and future critical control applications. The results show that our approach generates extensible schedules which enables Fog nodes to handle Fog applications with a shorter response time and a larger number of future control applications.

1. Introduction

Fog Computing, as an architectural means to realize the Operational Technology (OT)/Information Technology (IT) convergence [1], has emerged as a promising paradigm for enabling applications in various domains such as connected vehicles [2] and the Industrial IoT (IIoT) [3]. Other terms (e.g., *Edge Computing*) with similar objectives and principles are also used for such platforms [4]. As depicted in Fig. 1, a Fog Computing Platform (FCP) includes nodes (shown with boxes in the figure) capable of communicating and executing computations that needs to integrate mixed-criticality applications (shown with "Apps" in the figure) with different timing- and safety-criticalities [2,5], i.e., Fog Nodes (FNs), in the proximity of the "things" (e.g., machines) and data sources [6] to guarantee effective collaboration between the devices, nodes, and the Cloud.

This paper assumes the industrial automation has been implemented using an FCP consisting of heterogeneous FNs with different capabilities, such as [7–9]. The high-end FNs have high computation, communication and storage capabilities, whose resources can be shared by critical and non-critical applications [10]. We also assume that the FCP uses IEEE 802.1 Time-Sensitive Networks (TSN) [11] as a deterministic communication solution, as envisioned by several industrial consortia [1]. TSN consists of a set of amendments to the IEEE 802.1 Ethernet standard to provide features useful for real-time and safety critical applications. TSN supports multiple traffic types, and hence, is suitable for mixed-criticality applications running on an FCP.

At one extreme, we have the safety-critical real-time applications that control industrial process, have to be operational even in the case of failure, and need guaranteed control performance captured via Quality-of-Control (QoC) metrics. Due to their safety nature, such applications have to be certified. Certification standards require that safety functions of different criticality levels are protected (isolated), so they cannot influence each other [12], and the resources needed for their operation have to be statically assigned pre-release. Any changes in their functionality or configuration will trigger a costly re-certification. We call these applications "critical control applications". They are implemented using periodic hard real-time tasks running on FNs and messages (transmitted as flows on the network) and they are allocated resources via a static design-time configuration on the FNs and the network.

At the other extreme, we have non-critical dynamic applications that do not have stringent timing requirements. Such "Fog applications" implement the innovative functionality needed to realize Industry 4.0 on the converged OT/IT Fog infrastructure, without jeopardizing the performance and safety of the critical control applications. These

* Corresponding author.
E-mail addresses: mohba@dtu.dk (M. Barzegaran), paupo@dtu.dk (P. Pop).

https://doi.org/10.1016/j.sysarc.2022.102776
Received 23 May 2022; Received in revised form 23 September 2022; Accepted 25 October 2022
Available online 2 November 2022

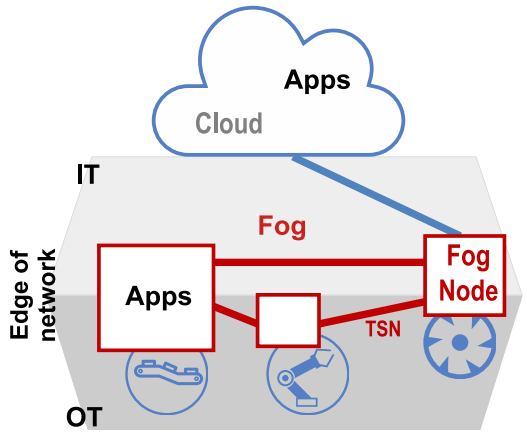


Fig. 1. Fog Computing Platform: Fog Nodes (FNs, depicted with boxes), equipment and the Cloud are connected via network (thick lines). Mixed-criticality applications ("Apps") are running on FNs and in the Cloud [13].

Fog applications are aperiodic, i.e., their arrival-time is unknown, but the FCP should dynamically allocate resources such that their quality-of-service (QoS) is maximized, e.g., their response times are reduced.

Fig. 2(a) shows an example industrial use case with several robots that use an FCP architecture. The critical control applications are responsible for the operation of robots, whose tasks and flows are implemented on the FNs and TSN, respectively. The configuration for these tasks and flows is determined at design-time such that it ensures the correct functionality of the critical control applications. We consider that the system engineers would like to deploy a set of innovative Fog applications for data analytics to optimize the industrial processes. They are deployed on the FCP via one of the FNs or from the Cloud. Also, we assume that at some time in the future, there is a need for extending the robot control, e.g., via an updated pathfinder algorithm, see Fig. 2(b). To minimize safety assurance costs, the new robot control should be added to the FCP without modifying the existing configuration, since modifications may trigger a costly re-certification. In addition, the Fog applications must run on the FCP at runtime without modifying the statically defined configuration for the control tasks and flows. Therefore, the static configuration should be extensible: resources for possible extensions of critical control applications have to be added pre-release and the resources that Fog applications use at runtime should be allocated at design-time.

Problem Definition: In this paper, we address the configuration of an FCP to support both critical control applications and Fog applications. We consider these two extremes because they will drive solutions that can support a wider range of applications. As we discuss in Section 6 that covers the related work, researchers have proposed a plethora of methods for the configuration of, on the one hand, critical control applications [5], and, on the other hand, Fog applications [14]. The state-of-the-art is to over-provision resources for critical control applications. These extra resources are often unused and wasted. Furthermore, it is costly to allow the addition of new safety functions without modifying the existing configurations by using the extra resources. However, their joint configuration has received limited attention [15] and the extensibility has not been addressed yet for FCPs that also need to accommodate Fog applications at runtime.

We consider that the critical control applications are configured at design-time and the Fog applications are handled via scheduling servers and configured at runtime using migration mechanisms which will decide the mapping of tasks and flows to the resources of the FCP available after the configuration of control applications. The FCP configuration consists of: (i) the mapping of critical control tasks to the cores of the FNs, (ii) the routing of critical control flows, (iii) the schedule tables for critical control tasks and flows, (iv) the slack in these schedule tables to increase their flexibility, and (v) the period and budget of the scheduling servers that allocate resources at runtime to the Fog applications. We formulate an optimization problem for the FCP configuration, such that (1) the deadlines and QoC of control applications are guaranteed at design-time, (2) the design-time FCP configuration is extensible and supports the addition of a larger number of future new control applications, and (3) the design-time configuration together with the runtime Fog resource management mechanisms, can successfully accommodate responsive Fog applications.

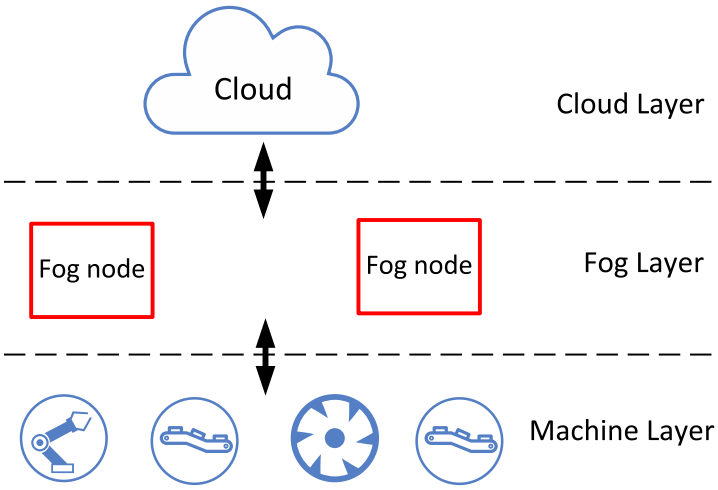
Contributions: The contributions of this paper are as follows. We motivate the need for supporting dynamic changes in an FCP to reduce the re-certification costs and increase productivity and the need for a novel FCP configuration optimization approach for mixed-criticality applications. Although the vision is to use partitioning to enforce the spatial and temporal isolation between applications with different criticalities in FCP, we ignored separation in this paper. However, our solution can be extended to include separation. We use a hierarchical scheduling model (Section 2.4) that can accommodate multiple scheduling policies, targeting the different time-criticality requirements of applications. The critical control applications are statically scheduled using static cyclic scheduling (i.e., they are time-triggered) and the resources of the Fog applications are allocated at runtime via fixed-priority servers that are dimensioned at design-time jointly with the critical application configurations. We consider that the critical control applications use Scheduled Traffic (ST) for their flows, implemented via IEEE 802.1Qbv, which defines a Time-Aware Shaper (TAS) mechanism that enables the scheduling of flows based on a global schedule table. The flows of the Fog applications use Strict Priority (SP) flows that are sent with lower priority in the gaps of the ST traffic schedule tables.

We formulate an optimization problem for the joint configuration of critical control and Fog applications, such that (1) the deadlines and QoC of control applications are guaranteed at design-time, (2) the design-time FCP configuration is extensible and supports the addition of a larger number of future new control applications, and (3) the design-time configuration together with the runtime Fog resource management mechanisms, can successfully accommodate responsive Fog applications.

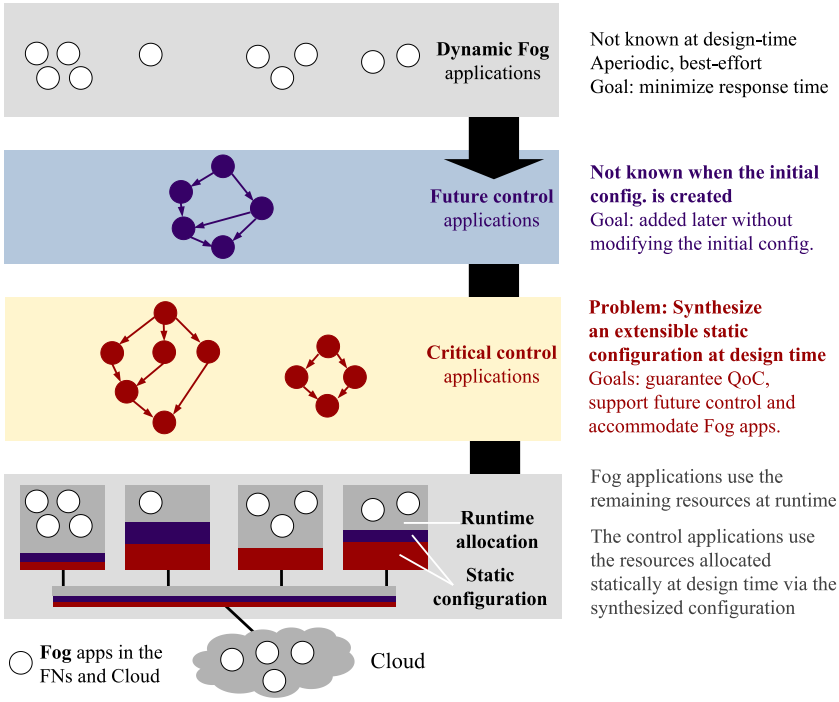
We propose a Constraint Programming (CP)-based optimization strategy to synthesize the optimized FCP configuration. Synthesizing a design-time configuration means deciding: (i) the mapping of critical control tasks to the cores of the FNs, (ii) the routing of critical control flows, (iii) the schedule tables for critical control tasks and flows, (iv) the slack in these schedule tables to increase their flexibility, and (v) the period and budget of the fixed-priority servers that allocate resources at runtime to the Fog applications. At runtime, our approach handles (vi) the migration of Fog tasks to the FNs that have resources for their execution, (vii) the scheduling of Fog tasks on the servers and of flows on TSN. We evaluate our CP approach on several synthetic and realistic test cases.

The novelty of this paper is the optimization solution which generates design time configuration of FCP to statically allocate resources for dynamical changes. These dynamical changes are in the form of future critical control applications and Fog applications. The allocated resources are used by fixed-priority servers to accommodate dynamic applications. The proposed solution optimizes the dimension of the allocated resources for providing better QoS for the dynamic applications.

The rest of the paper is structured as follows. Section 2 presents our underlying system model, and Section 3 presents the problem formulation. Afterwards, we present our configuration approach in Section 4, and evaluate the performance of our proposed approach in Section 5. Finally, Section 6 describes the related work, and Section 7 concludes the paper.



(a) Example use case with an FCP hosting mixed-criticality applications.



(b) Optimization problem overview

Fig. 2. An FCP hosting Fog applications and future control applications at runtime.

2. System models

The FCP model is presented in Section 2.1, and in Section 2.2 we present how TSN works for scheduled and strict priority traffic. The critical control and Fog application models are presented in Section 2.3 and Section 2.4 presents the scheduling policies used to jointly schedule these mixed-criticality applications. The summary of the notation is presented as an appendix.

2.1. Architecture model

We mode the architecture as a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \mathbf{ES} \bigcup \mathbf{SW}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. We model an architecture node as a vertex $v_i \in \mathcal{V}$ in the network graph. A node in the architecture is either an end-system (ES) (e.g., an FN, a sensor, an actuator, or a machine), or a network switch (SW). An ES is either the source (talker) or the destination (listener) of an application flow, whereas an SW forwards the frames of flows. Each architecture node has a set of input (ingress) ports and a set of output (egress) ports (denoted with $v_i.P$). A port $p_j \in v_i.P$ is linked to at most one other node. We denote the set of all FNs with $\mathcal{N} \subset \mathcal{V}$. Each FN $N_i \in \mathcal{N}$ has a multicore processor, and each core is denoted by $C_j \in N_i.C$.

We model the architecture bi-directional full-duplex physical links as a set edges $\mathcal{E}$ in the network graph. Thus, a full-duplex link between the nodes v_i and v_j is denoted with both $\epsilon_{i,j} \in \mathcal{E}$ and $\epsilon_{j,i} \in \mathcal{E}$; a link is attached to one port of the node v_i and one port of the node v_j.

The link $\epsilon_{i,j}$ is defined with the tuple $\langle s, d \rangle$ denoting the speed of the link in Mbit/s and the propagation delay function of the link in ms. The propagation delay of a frame on a link $\epsilon_{i,j}.d$ is calculated based the physical medium and the link length.

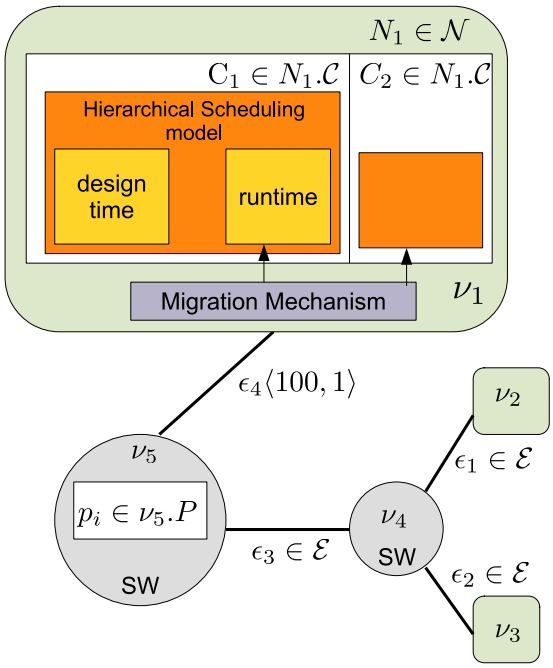


Fig. 3. Example architecture with three end-systems and two switches: The orange box shows the hierarchical scheduling model; the purple box shows the migration mechanism [16], all inside the node ν_1. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

A route $r_i \in \mathcal{R}$, where $\mathcal{R}$ is a set of routes, is an ordered list of links, starting with a link originating from a talker node, and ending with a link to a listener node. The number of links in the route r_i is denoted with $|r_i|$, and it starts from 2 since we assume there is at least one SW in the route. We define the function $\mathbf{u} : \mathcal{R} \times \mathbb{N}_0 \rightarrow \mathcal{E}$ to capture the j_{th} link of the route r_i.

An architecture model with three ESs (consisting of one FN and two sensors) and two SWs is presented in Fig. 3, where the thick lines are physical links. We also show in the figure examples on how the notation is used. The FCP is envisioned to host mixed-criticality applications which have to be separated from each other [7]. The separation is realized via temporal and spatial partitioning implemented via hypervisors [15] which has not been considered in this paper. However, separation can be introduced as additional constraints in our proposed CP model.

Additionally, the FCP is envisioned to handle Fog applications via monitoring and resource management techniques [7]. Once Fog applications are submitted to run on the FCP, the *Fog controller* FN, which is determined at runtime using mechanisms such as [17], receives the submission request. The Fog controller has knowledge on the available resources on the FNs and SWs using the resource discovery algorithms such as [17–19] at runtime. It can decide on the placement of application tasks on the FNs using a decentralized resource allocation technique [16,20] which determines the FN that provides the minimum response time for the application using response time analysis such as [21]. The fog controller also decides the routing of flows using a routing algorithm such as [22] considering the available resources on SWs at runtime. The scheduling policies used in the FCP are discussed in Section 2.4.

2.2. TSN switch model

Fig. 4 presents the architecture of a TSN switch. Switches are responsible for routing flows from input (ingress) ports to output (egress) ports. The flows are stored in the queues of egress ports before transmission. An egress port p_i has a set of (typically) eight priority queues $p_i.Q \in p_i.Q$, c.f. the IEEE 802.1Q standard [23].

We consider that the switches implement the 802.1Qbv standard, which uses "gates" for each queue and relies on a predefined Gate Control List (GCL) that controls the opening and closing time of the queue gates. With 802.1Qbv, the flows in a queue can be transmitted when a gate is open and cannot be transmitted when gate is closed. This requires a clock synchronization mechanism, e.g., 802.1ASrev [24] to define a "global" notion of time.

The flows of the critical control applications use Scheduled Traffic (ST) that is sent according to the predefined GCLs, and have higher priority. The flows of the Fog applications are sent in lower priority queues in the gaps of the ST traffic GCLs according to a Strict Priority (SP) FIFO scheme. Thus, ST flows use a subset of high priority queues, depicted with red in Fig. 4, and SP flows use a subset of lower priority queues, depicted with orange. The lowest priority queues (depicted with white) are for the best effort, which does not require any timing guarantees.

In this paper we assume that the GCLs are deterministic, i.e., the flows are isolated from each other: Only the frames of one of the flows are present in a queue at a time, see [25] for details. Flows of Fog applications are lower priority and are sent based on their priorities in the intervals of time (also called windows) when the respective queue gates are open, i.e., when ST flows are not scheduled.

To this end, we define a periodic window W_{p_j} on each port of the network nodes $p_j \in \nu_i.P$. Each window is characterized by the tuple $\langle c, t\rangle$ denoting the capacity (i.e., the length within a period) and period of the window in ms. Additionally, each window will have several instances which are referred to as window slices, in a hyperperiod H (which is the system cycle, see Section 2.3). We associate each window slice $W^j_{p_i}$ with its start time ϕ.

2.3. Application model

There have been several application models proposed in the literature, depending on the periodicity and time-criticality of the applications [26]. Our application model consists of (i) a set of critical control applications considered at design-time, denoted with Γ, which we capture using a periodic hard real-time task model, and (ii) a set of Fog applications considered at runtime, denoted with Γ', for which we use an aperiodic best-effort task model.

Critical control applications consist of tasks which exchange messages and implement control functions for controlling dynamical systems, see [15] for more details. All tasks and messages in a critical control application are periodic, can be of different periods [27], and have multiple periods [15]. However, without the loss of generality we assume that they have the same period. Thus, we define a hyperperiod H which is a system cycle and equal to the least common multiple of all application periods. Each critical control application $\gamma_i \in \Gamma$ is modelled with a directed acyclic graph (DAG), where a node represents either a task or a message, and edges represent data exchange between the nodes. The set of all tasks and the set of all flows in a critical control application are denoted with $\gamma_i.\mathcal{T}$ and $\gamma_i.\mathcal{F}$ respectively.

A critical task $\tau_j \in \gamma_i.\mathcal{T}$ is characterized by the tuple $\langle t, d, c\rangle$ denoting the task period, the task deadline and a known Worst-Case Execution Time (WCET) on the mapped FN in ms. 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 cores of FNs is captured by the function $\mathcal{M}$ which is determined by our proposed scheduling algorithm. The task τ_i will have $H/\tau_i.t$ instances denoted with $|\tau_i|$ in a hyperperiod H which are referred to as jobs denoted with τ^j_i. A job is associated with ϕ denoting the start time of the job.

A critical flow $f_i \in \gamma_i.\mathcal{F}$ is responsible for sending the frames that encapsulate the data from an application and it is characterized by the tuple $\langle p, c, t, d\rangle$ denoting the priority, the size in bytes, the period in ms, inherited from the originating task period, and the flow deadline,

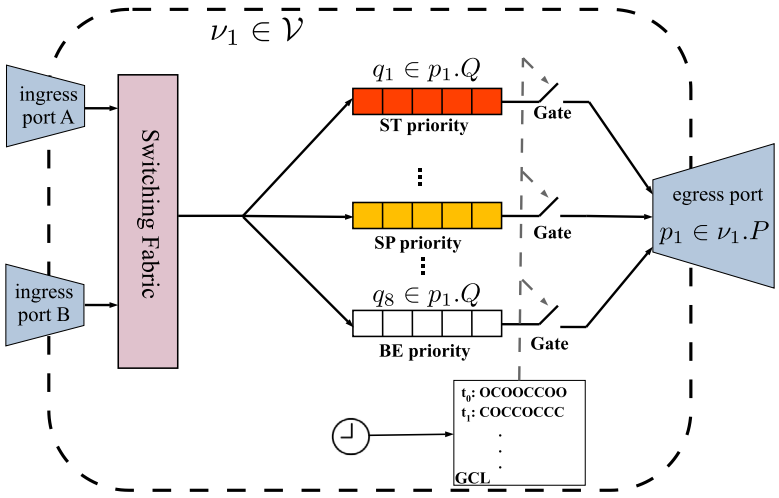


Fig. 4. TSN switch internals [13].

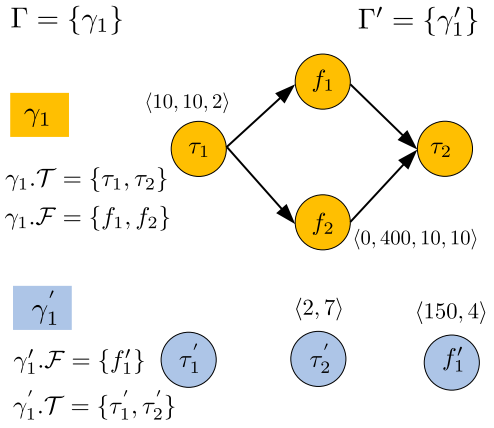


Fig. 5. Example application model with one critical control application and one Fog application.

i.e., the maximum allowed end-to-end delay in ms. Depending on the period, the frames of a flow will have to be transmitted multiple times within a hyperperiod, and we refer to each such transmission as an *instance* of a flow. The number of instances for a flow f_i is denoted with $|f_i|$, and is derived from the period of the flow t and the hyperperiod H.

Each flow f_i is transmitted via a route r_j which is captured by the function $\mathbf{z} : \mathcal{F} \rightarrow \mathcal{R}$ that maps the flows to the routes. We assume that each flow is associated to only one route, but several flows may share the same route. We also assume that the flows are unicast, i.e., there is only one listener for a flow. Our model can easily be extended to handle multicast flows, i.e., that have multiple listeners. We define a frame for each instance $1 \leq m \leq |f_i|$ of the flow f_i and on each link $1 \leq k \leq |r_j|$ of the route r_j, and denote it with $f^k_{i,m}$. A frame $f^k_{i,m}$ is associated with ϕ denoting the start time of the frame.

Each Fog application $\gamma'_i \in \Gamma$, consists of a set of aperiodic tasks and a set of aperiodic flows denoted by $\gamma'_i.\mathcal{T}$ and $\gamma'_i.\mathcal{F}$, respectively. The tasks do not have data dependencies, i.e., a task will start when it arrives, but they may exchange data asynchronously using flows. A Fog task $\tau'_j \in \gamma'_i.\mathcal{T}$ is denoted by the tuple $\langle c, j \rangle$ denoting the workload (which is the average execution time) on the mapped core of FNs in ms and its arrival time in ms. The arrival times and workloads of Fog tasks are unknown at design time. A Fog flow $f'_j \in \gamma'_i.\mathcal{F}$ is also aperiodic. Such a flow is denoted by the tuple $\langle c, j \rangle$ denoting the size in bytes and the arrival time in ms.

An example application model composed of two applications is shown in Fig. 5. γ_1 is a critical control application and γ'_1 is a Fog application. The task and flow details are given in the figure.

2.4. Scheduling policies

Mixed-criticality applications require different scheduling policies depending on their timing criticality [28]. Similar to related work, we use static cyclic scheduling (timeline scheduling) for critical control applications, since this is a scheduling policy suited for hard real-time applications in safety-critical areas. To put together several scheduling policies we use the hierarchical scheduling model [29], which consists of several levels of schedulers, starting at level 1 where a single scheduler reserves resources to applications and schedulers at the next level, i.e., level 2 [30]. Because the hierarchical scheduling model is general, we can accommodate any combination of schedulers needed by the mixed-criticality applications, in-between the two extremes: the static scheduling of critical control applications versus the dynamic scheduling of Fog applications.

Our scheduling model employs a static scheduler at level 1 for scheduling control applications and a periodic fixed priority server at level 2. The scheduler at level 1 uses a static cyclic scheduling, also known as time-triggered or timeline scheduling [28]. A static cyclic schedule captures the start and finishing time of tasks and flows and repeats with the hyperperiod H.

A fixed priority server is implemented as a periodic task D_{C_i} that runs in a core C_i and it is characterized by the tuple $\langle c, t \rangle$ denoting the capacity of the server (i.e., the length of the server within a period) and the period of the server in ms. Within a hyperperiod H, a server will have several instances which are referred to as server slices. Since level 1 uses static cyclic scheduling, the servers have to be scheduled in the static schedule at design-time together with the control tasks. Each server slice is denoted by $D^j_{C_i}$ and is associated with ϕ denoting its start time.

We consider that the aperiodic Fog tasks are served by such servers, and we use a deferrable server [31], which uses soft resource reservation techniques to allocate its resources to the Fog tasks. The servers serving aperiodic Fog tasks will use the slack introduced in the task schedule tables. We refer to the slack with the same notation used for the fixed priority server, and use them interchangeably.

Fog application flows are transmitted using the TSN mechanisms specific for SP flows in the windows when ST critical flows are not scheduled, see Section 2.2. These windows are similar to the slack in flow schedule tables.

3. Problem definition

We formally define the extensibility-aware configuration problem we address in the paper as follows, see Fig. 2(b) for an illustration. Given (1) a set of critical control applications Γ, (2) an FCP modelled with an architecture graph $\mathcal{G}$, we want to determine a configuration Ψ consisting of: (i) the mapping $\mathcal{M}$ of critical control tasks to the cores of the FNs, (ii) the set of routes $\mathcal{R}$ of critical control flows, (iii) the static task schedule tables, (iv) the GCLs, (v) the period and capacity of port windows W_{p_i}, and (vi) the period and capacity of the deferrable servers D_{C_i}. At runtime, our approach handles (vii) the migration of Fog tasks to the FNs that have resources for their execution, (viii) the scheduling of Fog tasks on the servers and of their flows on TSN.

We are interested in an optimized configuration Ψ such that: the deadlines of all the critical control applications are met, the QoC of control applications, as defined in Section 4.3, is maximized, and the extensibility of the configuration Ψ is maximized, which supports adding future control applications without modifications to Ψ and enables shorter response times for Fog applications.

Given a mapping $\mathcal{M}$, a static task schedule for the cores of FNs includes (i) the critical control task offsets $\tau_i^j.\phi$, and (ii) the server slices' offsets $D_{C_i}^j.\phi$. Additionally, GCLs for the ports of network nodes includes (i) the critical frames' offsets $f_{i,m}^k.\phi$, and (ii) the window offsets $W_{p_i}^j.\phi$.

4. Proposed solution

In this section, first, we describe our configuration optimization approach in detail and present an overview of the objective, a valid solution, and the extensibility metrics. Afterwards, we present the CP model and the CP constraints we use to solve the problem, in Sections 4.1 4.2, respectively. Finally, we define the objective function in Section 4.3 and present our heuristic search methodology in Section 4.4.

Just the scheduling problem, in even simpler contexts, has been proved to be Non-deterministic Polynomial time (NP)–complete in the strong sense [32,33]. We propose an optimization strategy called Extensible Configuration Optimization Strategy (ECOS), based on a CP formulation that uses search heuristics inside the CP solver, aiming at finding solutions even for large problem sizes. Fig. 6 presents an overview of ECOS which takes as the inputs the architecture model, and the application model; and outputs a set of the best solutions found during search.

As mentioned, ECOS is based on a CP formulation. CP is a declarative programming paradigm that has been widely used to solve a variety of optimization problems such as scheduling, routing, and resource allocations. With CP, a problem is modelled through a set of variables and a set of constraints. Each variable has a finite set of values, called domain, that can be assigned to it (see Section 4.1). Constraints restrict the variables' domains by bounding them to a range of values and defining relations between the domains of different variables (see Section 4.2). The search aims to find good quality solutions in a reasonable time, but it does not guarantee the optimality.

ECOS visits solutions that satisfy the constraints defined in Section 4.2 which are all valid solutions; and evaluates them using the objective function defined in Section 4.3 to check if the solution is an *improving* solution, i.e., better than the best solutions found so far. The objective function introduces metrics for extensibility and QoC. In an extensible solution, the slack is distributed in a way that is periodic, uniform and starts early within a period. Moreover, in a good QoC solution, the input–output jitter of a critical control application, i.e., variation in the end-to-end response of the application, is minimized.

By default, the CP solver systematically performs an exhaustive search by exploring all the possibilities of assigning different values to the variables. However, such a search is intractable for NP-complete problems, therefore we instead employ a metaheuristic which speeds up the search and helps finding the optimal solution in a shorter time, see Section 4.4.

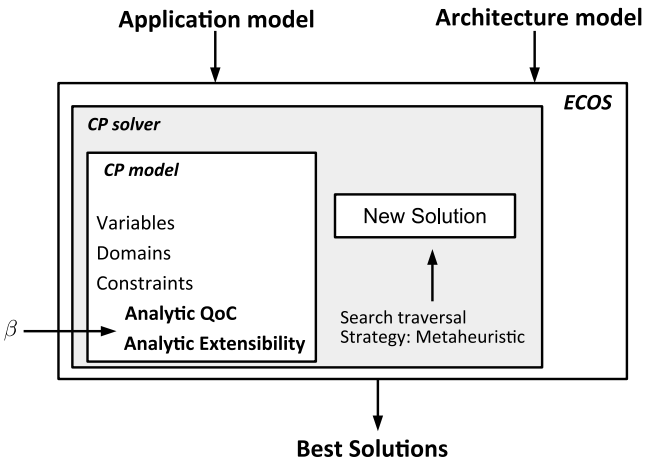


Fig. 6. Overview of ECOS.

4.1. CP model

We define seven sets of decision variables for the CP model. Each decision variable is associated with a domain from which the CP solver decides the variable's value. The decision variables and their domain for flows and windows are defined by

$$\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall m \in [1, \dots, |f_j|],\\
&\forall l \in [1, \dots, |r_k|], r_k = \mathbf{z}(f_j), e_{v,w} = \mathbf{u}(r_k, l) :\\
&0 \le f_{j,m}^k.\phi \times e_{v,w}.s \le (f_j.t \times e_{v,w}.s - f_j.c)\\
&\forall v_i \in \mathcal{V}, \forall p_j \in v_i.P, \forall m \in [1, \dots, H/W_{p_j}.t] :\\
&0 \le W_{p_j}^m.\phi \le (W_{p_j}.t - W_{p_j}.c)
\end{aligned} \tag{1}$$

where, the frame offsets' domain is the range of 0 to the time point when the frame has enough time to be transmitted within the frame period. The domain for window slice offsets is in the range of 0 to the time point when the slice accesses its capacity within its period.

For task scheduling, the decision variables are associated with the job offsets and the server slice offsets on cores of FNs. The variables and their domains are defined in Eq. (2) where the job offsets' domain is in the range from 0 to the time point when a duration equal to job's WCET is left within its period. Similarly, the domain for server slice offsets is in the range of 0 to the time point when the slice accesses its capacity within its period.

$$\begin{aligned}
&\forall \gamma_i \in \Gamma, \forall \tau_j \in \gamma_i.\mathcal{T}, \forall m \in [1, \dots, |\tau_i.t|] :\\
&0 \le \tau_j^m.\phi \le (\tau_j.t - \tau_j.c)\\
&\forall N_i \in \mathcal{N}, \forall C_j \in N_i.C, \forall m \in [1, \dots, H/D_{C_j}.t] :\\
&0 \le D_{C_j}^m.\phi \le (D_{C_j}.t - D_{C_j}.c)
\end{aligned} \tag{2}$$

The mapping function $\mathcal{M}$ which captures the mapping of tasks to the cores of FNs is also defined as a decision variable. The domain and the co-domain of the function is defined in Eq. (3) where the function domain is the set of all tasks in the application model and the function co-domain is the set of all cores of FNs.

$$\begin{aligned}
&\mathcal{M} : X \longrightarrow Y :\\
&X = \{\tau_i | \tau_i \in \gamma_j.\mathcal{T}, \gamma_j \in \gamma\} \quad Y = \{C_i | C_i \in N_j.C, N_j \in \mathcal{N}\}
\end{aligned} \tag{3}$$

4.2. CP constraints

Flow Constraints: We define five constraints similar to [13] that regulate the network traffic and relate the domain of the CP variables. CP only finds the feasible solutions, i.e., those where all the constraints are met.

The *Link Overlap constraint* does not allow sharing link resources with more than one frame or slice at a time, see Eq. (4) for the definition. As already mentioned in Section 2.1 a port is equivalent to a link, since a link is attached to only one source port and one destination port.

$$\begin{aligned}&\forall \gamma_i, \gamma_j \in \Gamma, \forall f_k \in \gamma_i.\mathcal{F}, \forall f_l \in \gamma_j.\mathcal{F}, k \neq l,\\&\forall m \in [1, \dots, |f_k|], \forall n \in [1, \dots, |f_l|],\\&r_o = \mathbf{z}(f_k), \forall q \in [1, \dots, |r_o|], \quad r_p = \mathbf{z}(f_l), \forall x \in [1, \dots, |r_p|],\\&p_{v,w} \equiv \epsilon_{v,w} = \mathbf{u}(r_o, q) = \mathbf{u}(r_p, x),\\&\forall y \in [1, \dots, H/W_{p_{v,w}}.t] :\\&((f^q_{k,m}.\phi + m \times f_k.t \geq f^x_{l,n}.\phi + n \times f_l.t + \frac{f_l.c}{\epsilon_{v,w}.s}) \vee\\&(f^x_{l,n}.\phi + n \times f_l.t \geq f^q_{k,m}.\phi + m \times f_k.t + \frac{f_k.c}{\epsilon_{v,w}.s})) \wedge\\&((f^q_{k,m}.\phi + m \times f_k.t \geq W^y_{p_{v,w}}.\phi + y \times W_{p_{v,w}}.t + W_{p_{v,w}}.c) \vee\\&(W^y_{p_{v,w}}.\phi + y \times W_{p_{v,w}}.t \geq f^q_{k,m}.\phi + m \times f_k.t + \frac{f_k.c}{\epsilon_{v,w}.s}))\end{aligned} \tag{4}$$

The *Route constraint* is defined to ensure that a frame is propagated from its talker to its listener via its associated route. As defined in Eq. (5), δ is the network precision that is the worst-case difference between the nodes clock in the network according to the 802.1AS clock synchronization mechanism [24].

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall m \in [1, \dots, |f_j|], \forall k \in [1, \dots, |r_l|),\\&r_l = \mathbf{z}(f_j), \epsilon_{v,w} = \mathbf{u}(r_l, k), \epsilon_{w,x} = \mathbf{u}(r_l, (k+1)) :\\&f^{k+1}_{j,m}.\phi \geq f^k_{j,m}.\phi + \frac{f_j.c}{\epsilon_{v,w}.s} + \epsilon_{v,w}.d + \delta.\end{aligned} \tag{5}$$

The *Isolation constraint*, as defined in Eq. (6) ensures displacements of frames in switches. The constraint checks frames with the same priority arriving at an ingress port of a switch at the same time. The constraint allows either a frame to arrive after or before any other frames on the same link, or frames having different priority on the same link to arrive at the same time. See [25] for more details on the order of frame transmission in the switch schedules. The δ in Eq. (6) presents the network precision.

$$\begin{aligned}&\forall \gamma_i, \gamma_j \in \Gamma, \forall f_k \in \gamma_i.\mathcal{F}, \forall f_l \in \gamma_j.\mathcal{F}, k \neq l,\\&\forall m \in [1, \dots, |f_k|], \forall n \in [1, \dots, |f_l|],\\&r_o = \mathbf{z}(f_k), \forall q \in (1, \dots, |r_o|], \quad r_p = \mathbf{z}(f_l), \forall x \in (1, \dots, |r_p|],\\&\epsilon_{v,w} = \mathbf{u}(r_o, q) = \mathbf{u}(r_p, x),\\&\epsilon_{a,v} = \mathbf{u}(r_o, q-1), \epsilon_{b,v} = \mathbf{u}(r_p, x-1) :\\&((f^q_{k,m}.\phi + m \times f_k.t + \delta \leq f^{x-1}_{l,n}.\phi + n \times f_l.t + \epsilon_{b,v}.d) \vee\\&(f^x_{l,n}.\phi + n \times f_l.t + \delta \leq f^{q-1}_{k,m}.\phi + m \times f_k.t + \epsilon_{a,v}.d)) \vee\\&(f_k.p \neq f_l.p).\end{aligned} \tag{6}$$

The *Flow Deadline constraint* is defined in Eq. (7) to ensure that a flow in delivered to its listener before its deadline is passed. To this end, the time interval between the transmission of a frame from its talker and the reception of the frame by its listener must be smaller than its deadline.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall m \in [1, \dots, |f_j|], r_n = \mathbf{z}(f_j),\\&\epsilon_{a,b} = \mathbf{u}(r_n, 1), \epsilon_{y,z} = \mathbf{u}(r_n, |r_n|) :\\&f^1_{j,m}.\phi + f_j.d \geq f^{|r_n|}_{j,m}.\phi + \frac{f_j.c}{\epsilon_{y,z}.s}.\end{aligned} \tag{7}$$

Task Constraints: We define three constraints for task scheduling on cores of FNs in the architecture model which relates the domain of the CP variables. A feasible solution meets all the constraints.

The *Core utilization constraint* avoids over utilization of cores. The constraint is defined in Eq. (8) where the utilization of all tasks and the server mapped to the same core is calculated.

$$\begin{aligned}&\forall N_i \in \mathcal{N}, \forall C_j \in N_i.C, \mathcal{T} = \{\tau_k | \mathcal{M}(\tau_k) = C_j, \tau_k \in \gamma_l, \gamma_l \in \Gamma\} :\\&\frac{D_{C_j}.c}{D_{C_j}.t} + \sum^{\tau \in \mathcal{T}} (\frac{\tau.c}{\tau.t}) \leq 1\end{aligned} \tag{8}$$

The *Task Overlap constraint* imposes the restriction on the solution not to allow a core to run more than one job or server slice at a time. The constraint is defined in Eq. (4).

$$\begin{aligned}&\forall \gamma_i, \gamma_j \in \Gamma, \forall \tau_k \in \gamma_i.\mathcal{T}, \forall \tau_l \in \gamma_j.\mathcal{T}, k \neq l,\\&\forall m \in [1, \dots, |\tau_i.t|], \forall n \in [1, \dots, |\tau_i.t|],\\&C_o = \mathcal{M}(\tau_k) = \mathcal{M}(\tau_l), \forall y \in [1, \dots, H/D_{C_o}.t] :\\&((\tau^m_k.\phi + m \times \tau_k.t \geq \tau^n_l.\phi + n \times \tau_l.t + \tau_l.c) \vee\\&(\tau^n_l.\phi + n \times \tau_l.t \geq \tau^m_k.\phi + m \times \tau_k.t + \tau_k.c)) \wedge\\&((\tau^m_k.\phi + m \times \tau_k.t \geq D^y_{C_o}.\phi + y \times D_{C_o}.t + D_{C_o}.c) \vee\\&(D^y_{C_o}.\phi + y \times D_{C_o}.t \geq \tau^m_k.\phi + m \times \tau_k.t + \tau_k.c))\end{aligned} \tag{9}$$

The *Deadline constraint* defined in Eq. (10) imposes the restriction that a job produces its outputs within its deadline.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall \tau_j \in \gamma_i.\mathcal{T}, \forall m \in [1, \dots, |\tau_j.t|] :\\&\tau^m_i.\phi + \tau_i.c \leq \tau_i.d.\end{aligned} \tag{10}$$

Application Constraints: We define four constraints for handling the data dependency between tasks and flows of an application which is modelled with a DAG, see Section 2.3. To this end, we define four functions capturing the task and flow precedence.

The function $\mathcal{J}^{\mathcal{T}}_{\tau} : \gamma.\mathcal{T} \longrightarrow \gamma.\mathcal{T}$ takes an application task as the input and returns a set of tasks which have precedence over the input task within the same application. Similarly, the function $\mathcal{J}^{\mathcal{F}}_{\tau} : \gamma.\mathcal{T} \longrightarrow \gamma.\mathcal{F}$ takes an application task as the input and returns a set of flows, the function $\mathcal{J}^{\mathcal{T}}_{f} : \gamma.\mathcal{F} \longrightarrow \gamma.\mathcal{T}$ takes an application flow as the input and returns a set of tasks, and the function $\mathcal{J}^{\mathcal{F}}_{f} : \gamma.\mathcal{F} \longrightarrow \gamma.\mathcal{F}$ takes an application flow as the input and return a set of flows, all within the same application.

The *Task Precedence over Task constraint* is defined in Eq. (11) and enforces that in an application and within a period, each task is executed after all the tasks determined by the function $\mathcal{J}^{\mathcal{T}}_{\tau}$ produce their outputs.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall \tau_j \in \gamma_i.\mathcal{T}, \forall \tau_k \in \mathcal{J}^{\mathcal{T}}_{\tau}(\tau_j),\\&\forall l \in [1, \dots, |\tau_j.t|], \forall m \in [1, \dots, |\tau_k.t|] :\\&\tau^l_j.\phi \geq \tau^m_k.\phi + \tau_k.c\end{aligned} \tag{11}$$

The flows' precedence over each task is regulated using the *Flow Precedence over Task constraint* defined in Eq. (12). The constraint avoids executing each task before all the flows decided by the function $\mathcal{J}^{\mathcal{F}}_{\tau}$ have arrived, within a period.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall \tau_j \in \gamma_i.\mathcal{T}, \forall f_k \in \mathcal{J}^{\mathcal{F}}_{\tau}(\tau_j),\\&\forall l \in [1, \dots, |\tau_j.t|], \forall m \in [1, \dots, |f_k.t|],\\&r_o = \mathbf{z}(f_k), \epsilon_{a,b} = \mathbf{u}(r_o, |r_o|) :\\&\tau^l_j.\phi \geq f^{|r_o|}_{k,m}.\phi + \frac{f_k.c}{\epsilon_{a,b}.s}\end{aligned} \tag{12}$$

Similarly, the *Task Precedence over Flow constraint* defined in Eq. (13) regulates the precedence of tasks determined by the function $\mathcal{J}^{\mathcal{T}}_{f}$ over each flow. Upon the output production of all determined tasks, the flow is scheduled for transmission.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall \tau_k \in \mathcal{J}^{\mathcal{T}}_{f}(f_j),\\&\forall l \in [1, \dots, |f_j.t|], \forall m \in [1, \dots, |\tau_k.t|],\\&r_o = \mathbf{z}(f_j), \epsilon_{a,b} = \mathbf{u}(r_o, 1) :\\&f^1_{j,l}.\phi \geq \tau^m_k.\phi + \tau_k.c\end{aligned} \tag{13}$$

Eq. (14) defines the *Flow Precedence over Flow constraint* which regulates the transmission of a flow that is data dependent to a set of flows determined by the function $\mathcal{J}_f^F$.

$$\begin{aligned}&\forall \gamma_i \in \Gamma, \forall f_j \in \gamma_i.\mathcal{F}, \forall f_k \in \mathcal{J}_f^F(f_j),\\&\forall l \in [1,\dots,|f_j.t|], \forall m \in [1,\dots,|f_j.t|],\\&r_n = \mathbf{z}(f_j), r_o = \mathbf{z}(f_k),\\&\epsilon_{a,b} = \mathbf{u}(r_n,1), \epsilon_{v,w} = \mathbf{u}(r_o,|r_o|) :\\&f_{j,l}^1.\phi \geq f_{k,m}^{|r_o|}.\phi + \frac{f_k.c}{\epsilon_{v,w}.s}\end{aligned} \tag{14}$$

4.3. Objective function

The CP solver propagates the constraints all over the search space and removes the unfeasible solutions (which do not satisfy the constraints) from the search space that results in the creation of the solution space. Afterwards, the CP solver picks the first solution from the solution space and determines the value of the objective function for the solution. The CP solver searches for better solutions in terms of the objective function until no such solutions can be found.

We define the objective function Ω in Eq. (15). The terms Θ_1 and Θ_2 in the equation capture the analytical QoC and the extensibility, respectively. The weight β controls the trade-off between QoC and extensibility towards finding a solution with either a better QoC or a shorter response time for Fog applications. The weight β is between 0 to 2, where a larger β drives the search towards response time optimized solutions.

$$\Omega = \Theta_1 + \beta \times \Theta_2^{-1} \tag{15}$$

Analytical QoC CP model: In this work, we are interested in finding the solutions which have better QoC (see [34] for more information on the QoC and see [13,15] for more information on the schedule optimization for the QoC). Since calculating the QoC needs a simulation of the control application's behaviour and the integration of QoC calculation tools such as JitterTime in the CP model are not feasible due to large runtimes, we have adapted the analytical QoC model proposed in [13] and integrated it into our CP model.

The model formulates the QoC as: (i) minimum jitter for end-to-end input–output flows, (ii) maximum delay between reception of the input flow and transmission of the output flow, called task execution interval, and (iii) minimum jitter for the task execution interval.

The analytical QoC CP model is formulated in Eq. (16), where γ_i is a critical control application and the terms θ_1 captures the input flow delay, θ_2 the output flow delay, θ_3 models the input flow jitter, θ_4 the output flow jitter, and θ_5 the task execution interval jitter. The range of all the θ terms is from 0 for no delay/jitter to 1 for a delay/jitter equal to the control application's period. The delay vs. jitter trade-off is controlled by the weight β_1 which can direct the search towards either optimized delay or optimized jitter, concerning the type of the control applications. A larger β_1 value drives the search towards smaller jitter. As proposed in [13], the β_1 value is determined using JitterTime.

$$\begin{aligned}&\forall f_j \in \mathcal{J}_t^f(\tau_1), \forall f_k \in \gamma_i.\mathcal{F} \setminus \mathcal{J}_t^f(\tau_1), \tau_1 \in \gamma_i.\mathcal{T}\\&\forall m,q \in [1,\dots,|f_j|], \forall n,u \in [1,\dots,|f_k|],\\&r_o = \mathbf{z}(f_j), r_p = \mathbf{z}(f_k) :\\&\theta_1 = \sum \frac{f_{j,m}^{|r_o|}.\phi}{f_j.t} \quad \theta_2 = \sum \frac{f_k.t - f_{k,n}^1.\phi}{f_k.t}\\&\theta_3 = \sum \frac{|f_{j,m}^{|r_o|}.\phi - f_{j,q}^{|r_o|}.\phi + (m-q)\times f_j.t|}{f_j.t}\\&\theta_4 = \sum \frac{|f_{k,n}^1.\phi - f_{k,u}^1.\phi + (n-u)\times f_k.t|}{f_k.t}\\&\theta_5 = \sum \frac{|f_{k,m}^{|r_p|}.\phi - f_{k,q}^{|r_p|}.\phi + f_{j,q}^1.\phi - f_{j,m}^1.\phi + (m-q)\times f_j.t|}{f_j.t}\\&\Theta_1 = \theta_1 + \theta_2 + \beta_1 \times (\theta_3 + \theta_4 + \theta_5)\end{aligned} \tag{16}$$

Analytical extensibility CP model: This paper assumes using periodic slacks in the static task schedule for hosting Fog tasks and future critical control tasks; and periodic windows in queue gates for sending Fog flows and future critical control flows. We are interested in finding a distribution of the slacks and windows that provides a shorter response time for Fog applications and host a larger number of critical control applications. There are different techniques in the literature for analysis and determining such a distribution of the slacks and windows [21,35–37].

The typical technique is to first, calculate the required capacity, and then, determine the slack and window distribution that uses the least system resources and is able to provide the required capacity, at any time [38]. A common method for the calculation of the required capacity is the "submission load technique" [39]. Thus, we define a general *Load function* $\mathcal{L}(t)$ in Eq. (17) for the required capacity calculation at time t. In Eq. (17), the function $u(t-a)$ is a delayed unit step function, n is the total number of tasks and flows in an application, j_k is the arrival time of the tasks and the flows, and c_k is the required capacity, i.e., the task workload and the message transmission time.

$$\mathcal{L}(t) = \sum_{i=1}^{n}\sum_{k=1}^{i} u(t-j_k)\times c_k \tag{17}$$

There are also different techniques for calculating the reserved resources in the real-time theory such as "server characteristic function" proposed in [37], "server supply function" proposed in [40], and "availability function" proposed in [21]. We define the "Availability function" $\mathcal{A}(t)$ in Eq. (18), where the function $r(t-a)$ is a delayed unit ramp function, $S.t$ is the slack/window period, $S.c$ is the slack/window capacity, and the $S^i.\phi$ is the offset of the ith slack/window slice. The unit step and unit ramp functions are the most common functions in math and definition of a function delay is very well-known in math. Thus, there is no need to present their definition.

$$\begin{aligned}&\mathcal{A}(t) = \sum_{i=1}^{\lceil \frac{t}{S.t} \rceil} (r(t-\alpha_1) - r(t-\alpha_2))\\&\alpha_1 = j \times S.t + S^i.\phi \quad \alpha_2 = \alpha_1 + S.c\end{aligned} \tag{18}$$

Once the load function and the server availability function are known, the response time Res can be calculated using Eq. (19) as proposed in [21]. Eq. (19) can be solved using iterative procedure starting at $t = 0$ until Res^{n+1} and Res^n converge. As mentioned, for Fog applications we focus on finding the solutions that provide a shorter response time, and for future critical control applications solutions that can host a larger number of applications without missing their deadlines. Such an analysis can also be used for future critical control applications as long as the determined response time is smaller than the deadlines.

$$Res = \mathcal{A}^{inv}(\mathcal{L}(Res)) \tag{19}$$

However, the presented analysis and design approach cannot be employed at design-time since a prior knowledge on Fog applications and future critical control applications is not available (they arrive at runtime). To this end, we use the idea of the availability function in Eq. (18) and the extensibility metric presented in [41] to define an analytic extensibility CP model in Eq. (20), where $S.t$ is the slack/window period, $S.c$ is the slack/window capacity, and the $S^i.\phi$ is the offset of the ith slack/window slice. This analytic model calculates the accumulated reserved resources within a hyperperiod, capturing the distribution of the slack/window slices.

$$\begin{aligned}\Theta_2 &= \frac{\mathbf{n}\times S.c}{2} \times ((\mathbf{n}+1)\times S.t - S.c)\\&- \sum_{i=1}^{n} S.c \times S^j.\phi, \quad \mathbf{n} = \frac{H}{S.t}\end{aligned} \tag{20}$$

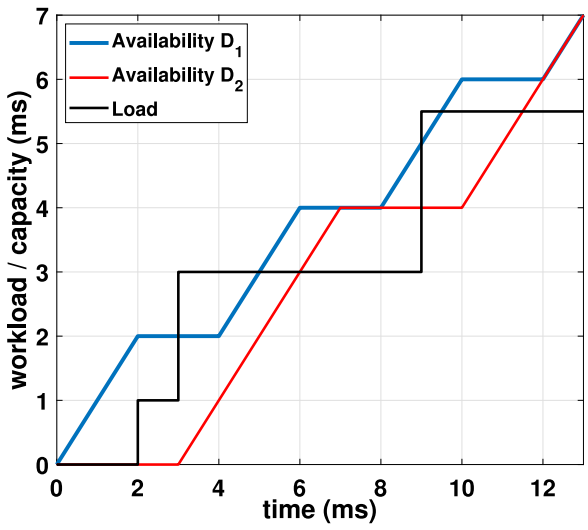


Fig. 7. Example server availability and load: The server D_1 (blue curve) provides a shorter response time comparing to the server D_2 (red curve) for the known application load (black curve). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

We give an example Fog task handling problem in Fig. 7 where the black curve is the load function, the blue curve is the server D_1 availability function, and the red curve is the server D_2 availability function. The load function is calculated for the application γ_1' consisting of three tasks specified as $\tau_1'\langle 1,2\rangle$, $\tau_2'\langle 2,3\rangle$, and $\tau_3'\langle 2.5,9\rangle$. The servers $D_1\langle 2,4\rangle$ and $D_2\langle 4,7\rangle$ have 7 and 4 server slices, respectively. For all slices of server D_1 and server D_2, the offsets are $D_1^j.\phi = 0$ and $D_2^j.\phi = 3$ ms.

We used the function in Eq. (19) to calculate the server's response time for the given tasks. Although the server D_2 has a higher utilization (57% comparing to 50% of the server D_1), it provides a loner response time for all tasks which is 3.5 ms (comparing to 3 ms of the server D_1). Thus, the server D_1 is a better choice as our proposed extensibility CP model approves. Our proposed extensibility CP model in Eq. (20) calculates the extensibility values of 210 and 200 for servers D_1 and D_2, respectively. As the example shows, our proposed analytic extensibility CP model has successfully determined the best server configuration, i.e., that provides the shorter response time.

4.4. Search strategy

In this work we used the Google OR-Tools [42] as the CP solver which is configured to use a *metaheuristic* as the search strategy. A search strategy specifies the order of selecting the decision variables for assignment and the order of selecting the values from the domain of a decision variable. The metaheuristic strategy does not guarantee optimality, but it is effective in finding good quality solutions in a reasonable time.

We used the same metaheuristic strategy as presented in [13,43] based on a Tabu Search metaheuristic algorithm [44], which aims to avoid the search process being trapped in a local optimum by increasing diversification and intensification of the search. We apply the metaheuristic strategy to all set of decision variables defined in the CP model. In this strategy, once a task or flow is scheduled with the respective minimum objective value, it is treated as keep variables whose values should not be changed.

5. Evaluation

The benefits of Fog Computing in industrial automation have been already evaluated via realistic demonstrator implementations [45,46]. In this section, we are interested in evaluating the ability of our design-time tool to create extensible configurations that guarantee the safety and real-time requirements for the critical control and support the addition of dynamic Fog applications at runtime. The evaluation is done using both a realistic use case and synthetic test cases, inspired from realistic industrial applications. This section is structured as follows: we first describe our evaluation setup and the test cases in Section 5.1. Afterwards, we evaluate our proposed ECOS method for its ability to support future critical control applications (Section 5.2) and to accommodate Fog applications (Section 5.3). Finally, we evaluate the performance of ECOS on a realistic test case where we consider scenarios of upgrades with future critical control applications and migrations of Fog applications (Section 5.5).

5.1. Test setup and scenarios

Our proposed ECOS approach is implemented in C# using Google OR-Tools [42] as the CP solver and it is run on a computer with an i9 CPU at 3.6 GHz and 128 GB of RAM. We limited the search in CP solver for a duration of 15 to 120 min depending on the test case size and the scenario. We assumed that all links have a speed of 1 Gbit/s.

We have generated ten synthetic test cases with critical control applications that have progressively larger number of tasks and flows, whose details are given in Table 1. The columns 2, 3, 4, 5 and 6 in the table show the total number of critical control applications, tasks, flows, ESs, and SWs for the test cases, respectively. Each critical control application implements a Linear–quadratic–Gaussian (LQG) control function which is designed with Jitterbug [47] for controlling plants in the form of Eq. (21) where a and b are randomly chosen respectively from $[50, 100, 150]$ and $[100, 200, 300, 400]$ (see [15] for more details).

$$G = \frac{a}{s^2 + b} \tag{21}$$

The critical control flows are generated randomly with message sizes to fit in single maximum transmission unit (MTU)-sized frames. The tasks are also generated with random WCETs. Tasks and flows of each critical control application have equal periods and deadlines in the form of 2^n ms, $n = \{0, 1, 2, 3, 4\}$. The column 7 in the table shows the mean utilization of critical control applications which is the average of tasks' CPU utilization and flows' bandwidth utilization.

5.2. Supporting future control applications

We were first interested to evaluate the ability of ECOS to generate configurations that allow updates, i.e., the addition of future critical control applications with no changes to the current configuration. For this purpose, we created for each test case in Table 1 a scenario where critical control applications have to be added in the future. As depicted in Table 2, columns 3 and 4 show for each scenario the total number of tasks/flows, and mean utilization of the applications, respectively.

Together with ECOS, we evaluated a version, called ECOS/E that does not optimize for extensibility, i.e., it does not consider the extensibility term Θ_2 in the cost function Ω and instead optimizes only for QoC term Θ_1. For the configurations obtained with ECOS and ECOS/E on each test case, we used again ECOS to add the future critical control applications for the respective test case, considering that the configuration of the initial critical control applications cannot be changed. This is to avoid re-certification, as discussed in Section 1.

Since an extensible configuration aims to support future critical control applications, i.e., their tasks and flows have no missed deadlines, we show in columns 5 and 6 of Table 2 the percentage of supported tasks and flows in ECOS and ECOS/E, respectively. The conclusion is that ECOS can better support future critical control applications by considering the extensibility of the configurations during their optimization. The main message is that without considering the extensibility, i.e., using ECOS/E, none of the future upgrades can be performed; all values in column 6 under are below 100%, showing

Table 1
Details of ten critical control (CC) synthetic test cases.

#	Total no. of CC apps	Total no. of CC tasks	Total no. of CC flows	Total no. of ESs	Total no. of SWs	Mean util. of CC apps
1	5	17	11	3	1	34%
2	7	20	14	5	1	39%
3	10	24	18	7	2	42%
4	12	26	20	9	2	44%
5	15	28	25	11	2	48%
6	18	30	30	13	3	51%
7	20	34	34	15	3	53%
8	23	36	38	17	3	56%
9	27	40	45	19	4	59%
10	30	42	45	21	4	61%

Table 2
Evaluation results on synthetic test cases.

TC[a]	Runtime[b]	Total no. of tasks/flows in FCCAs[c]	Mean util. of FCCAs[c]	Percentage of Supported FCCAs[c]		RT[d] of Fog application 1 Tasks: 16 Flows: 15		RT[d] of Fog application 2 Tasks: 21 Flows: 20		RT[d] of Fog application 3 Tasks: 35 Flows: 38	
#				ECOS	ECOS/E	ECOS	ECOS/E	ECOS	ECOS/E	ECOS	ECOS/E
1	3.18	36/58	57%	100%	78%	1.33	3.97	2.82	5.66	4.73	7.43
2	4.21	37/65	55%	100%	89%	1.42	1.91	2.75	4.92	6.74	9.56
3	6.88	30/20	50%	100%	96%	1.26	3.36	2.94	4.88	5.16	12.29
4	8.06	29/32	45%	100%	81%	2.17	3.27	3.64	5.65	5.18	7.34
5	9.73	33/40	44%	100%	96%	1.56	4.14	3.68	5.81	4.36	9.44
6	12.35	44/45	40%	100%	90%	1.47	2.96	3.14	4.17	4.96	5.82
7	13.92	37/35	39%	100%	83%	1.19	3.95	3.88	3.96	2.96	4.76
8	14.96	33/34	37%	100%	90%	2.18	2.93	3.14	5.84	2.84	9.64
9	16.38	25/28	31%	100%	98%	1.65	3.77	4.43	5.93	2.35	4.74
10	17.29	18/21	27%	100%	82%	2.37	3.79	4.82	5.97	2.85	6.68
		Average		100%	87%	1.6	3.4	3.5	5.2	4.2	7.7

[a]Test Case.
[b]Runtime in seconds.
[c]Future Critical Control Application.
[d]Response time in ms.

that the upgrades are not accommodated. As the results show, only ECOS can add all the future critical control application. In some cases, ECOS shows that it can support 23% more tasks and flows compared to ECOS/E. Thus, ECOS shows a good performance in supporting future control applications that avoids re-certification costs.

5.3. Response time analysis of fog applications

We were also interested to evaluate the ability of ECOS to accommodate Fog applications. Hence, we created for each test case in Table 1 a scenario with three sets of Fog applications that will need to be hosted by the FCP at the same time with the critical control applications. Each set of Fog applications consists of different number of tasks and flows with random arrival times, flow sizes, and task workloads. Their details are presented in columns 7 to 12 of Table 2. We simulate their migration to the FNs of the FCP considering the migration mechanism from Section 2.1. Fog flows in Table 2 are Best Effort and are using the SP traffic type, as discussed in Section 2.2.

Similar to the previous section, we evaluated ECOS against ECOS/E, where extensibility has not been considered. Thus, as in the previous section, we used ECOS and ECOS/E to generate optimized configurations for each test case. As discussed in Section 3, we are interested to host Fog applications with the shortest response times. ECOS uses the extensibility terms Θ_2 in the cost function Ω to favour solutions that are extensible, i.e., they have well-dimensioned servers and windows for Fog applications.

Columns 7 to 12 in Table 2 report the mean response time of Fog applications under the respective configuration using the Eq. (19) in ms. As we can see from Table 2, by considering the extensibility when creating the configurations at design-time, we are able to accommodate Fog applications at runtime with reduced response times with an average 43%. In some test cases, such configurations generated by ECOS could reduce the response time by 3.71 ms (see test case 3) and improve the response time by 51% (see test case 1). When ECOS/E is used, we can see that the response times for most Fog applications are much larger. That is because although the total resources available to the Fog applications are the same in both the ECOS and ECOS/E configurations, the deferrable servers and communication windows, which allocate the available resources to Fog applications, are not well dimensioned in the case of ECOS/E, and hence cannot be used at runtime. This shows the importance of configuring for extensibility in the Fog.

5.4. Runtime and complexity analysis

The problem addressed in this paper is interactable. ECOS, as a decision making solution, generates a configuration which consists of deciding on the schedule tables, which has been proved to be Non-deterministic Polynomial time (NP)–complete in the strong sense [32, 33]. Such solutions are exact optimization methods similar to Brunch & Bound, Cutting-plane, and Integer Linear Programming, which have exponential efforts. The longer the method runs, the better is the solution. However, ECOS can be asked to return a good quality solution within a time limit, thanks to the search methodology we employ in the Google OR tools (see Section 4.4). The time limit is decided based on the size of the problem such that finding a good solution is guaranteed.

We give the ECOS runtime for each test case in column 2 in Table 2. The runtime is reported for generating design-time configurations for critical control applications. Fog applications are handled at runtime using a fixed-priority server which is a simple cycle-based scheduler and has a low complexity. As the results show, we are able to find a good solution in a short time comparing to the given time limit for all test cases. In all test cases, the good solution is found in a less than 1% of the given time limit. The runtime increases with the test case sizes and it is in an acceptable range, based on the discussion with potential users.

Table 3
Fog-based pharmaceutical production line.

App set	Apps	No of Tasks/flows	Bandwidth util.	CPU util.
Critical control app	Agitator	6/5	0.7%	8%
	Boiler	3/4	0.4%	7%
	Chiller	5/7	0.7%	10%
	Coater	9/13	1.1%	11%
	Pulverizer	8/9	0.6%	12%
	Tablet printer	14/15	1.2%	18%
	Conveyor	15/21	1.5%	21
Future critical control apps	Heat exchanger	6/8	0.9%	11%
	Sifters	9/10	1.1%	17%
	Cartoners	7/10	0.8%	9%
	Sealer	10/12	0.9	9%
Fog apps	Label check	5/8	NA[a]	NA[a]
	Machine maintenance	12/13	NA[a]	NA[a]
	Air quality check	15/12	NA[a]	NA[a]
	Stock check	8/4	NA[a]	NA[a]
	Safety check	8/14	NA[a]	NA[a]

[a]Not Applicable.

5.5. Extending with upgrades

The next question we were interested to answer is if this ability of ECOS to accommodate Fog applications holds when we add future critical control applications. This scenario is adopted from industrial settings where upgrades in the form of future critical control applications are introduced while the ability of the FCP to host Fog applications are still needed. To this end, we present a realistic test case where a set of critical control applications as shown in Table 3 are considered at design-time. The test case consists of 4 ESs and 6 SWs implementing a pharmaceutical production line. We use ECOS to generate an extensible FCP configuration at design-time for these applications. In this configuration the mean utilization of critical control applications is 47%, and the FCP is able to host Fog applications (see their details in Table 3) with a response time of 3.6 ms.

Once a set of future critical control applications with a mean utilization of 26% are introduced as an upgrade (see Table 3), we again use ECOS to accommodate this upgrade. In the upgraded configuration, the slack which has been diminished for hosting Fog applications, since part of it has been assigned to run the future critical control applications. When adding these future control applications, ECOS has also optimized the new upgraded configuration for extensibility. As we can see, ECOS was able to update the configuration to still accommodate the Fog applications in Table 3, at the cost of a small increase in their response times, i.e., 1.76 ms. Note that in practice these response times will be smaller, because the critical control tasks will not execute for their WCET. Instead, they will finish earlier than WCET, and as we discussed in Section 2.4, the servers will use at runtime the freed computational resources.

6. Related work

There has been a lot of work in the literature for scheduling in mixed-criticality systems and several methods have been proposed, such as hierarchical scheduling [48], task partitioning and scheduling [49], container-based scheduling [50], and mixed traffic schedulers [51] for flows in particular. However, none of these works address the problem of extensibility in mixed-criticality systems, which is the focus of this paper.

Designing for evolvability and extensibility in computer systems has also been addressed in the literature [52,53]. Extensibility is defined as an ability that enables future upgrades and changes in computer systems [54]. These changes and upgrades can be implemented in different ways such as introducing new applications, migrating applications across processing nodes, and changes in required resources. An extensible system is designed in a way that accommodating changes and upgrades does not require re-designing and costly changes in the platform. Computer systems hosting safety-critical applications requires safety certification and any changes in the system configurations requires re-certification [55]. However, in this paper we focus on a form of extensibility that is accommodating future applications, thus, an extensible mixed-criticality that also hosts safety-critical applications, does not require re-certification.

Pop et al. [56] propose an incremental scheduling algorithm for embedded systems which aims at facilitating hard real-time applications implemented as tasks and adding specific future tasks. The approach generates extensible schedules that can accommodate future tasks without disturbing the existing tasks. This is realized using the slack in the schedule. The approach in [36] uses extensibility to target robust task scheduling in distributed systems. Any changes in the task requirements are considered an extension. The proposed robust scheduler is able to support these changes using the dimensionend slacks in the schedule. Zheng et al. [35] propose a mathematical modelling approach for extensibility. In this work extensibility is in the form of accommodating future tasks. This approach determines a distribution of the slack among all tasks and targets all variations of future task sets. A benefit of this approach is that no prior specification of future tasks is required. None of these works consider the communication aspect, which has a critical impact on extensibility, i.e., the computational elements of a distributed system has to share a network.

We proposed in [41] an extensible scheduling algorithm for critical applications in an FCP. The proposed algorithm employs a heuristic approach that provides well-distributed slacks in the schedules of high-critical applications, which can be used for scheduling future critical applications. In the current work, we take the network into account and consider the that the system uses TSN. The focus of previous work mentioned so far was on hard real-time tasks. In addition, we also consider realistic control tasks for which we take the QoC into account, and dynamic Fog applications.

Yin et al. [50] propose a task scheduling algorithm that targets extensibility by using resources in the Cloud. In this approach, tasks are scheduled either on FNs or in the Cloud using containers. In this method, the goal is to optimize the resource usage over the platform and make use of all resources.

The work in the literature also addresses extensibility in the systems hosting mixed-criticality network messages. [57] focuses on scheduling network messages in TSN networks in automotive where dynamic messages with less-criticality are needed to be scheduled with ones of high criticality (e.g., control engine); and optimizing the schedules to host more dynamic messages. Guo et al. [58] propose a method for mapping task to the control units of an automotive use case targeting extensibility. These tasks exchange messages in the automotive

network. However, the works aim at increasing the number of future messages hosted and do not consider optimizing the schedules for QoS or QoC.

The authors in [59] propose an approach which decides the mapping of applications to the processing elements, separates the mixed-criticality applications using partitioning, and schedules tasks and messages of the applications. This method aims for the extensible configuration which allows adding future applications. Similarly, the approach presented in [60], uses the worst-case response time analysis to decide on the task allocation, the signal to message mapping, and the assignment of priorities to tasks and messages. The approach optimizes the decisions for accommodating future applications. Compared to the related work, we propose an optimization approach for generating the extensible configurations at design time and mechanisms for handling future applications at runtime, considering mixed-criticality applications, such as hard real-time applications, control applications for which we optimize their QoC and dynamic Fog applications for which we optimize their QoS.

7. Conclusions

Fog Computing is an enabler for Industry 4.0 where mixed-criticality applications are running on a shared computing platform. The Fog Computing Platform (FCP) has to separate such applications to protect high-criticality applications. The vision is to virtualize control applications and run them as software tasks and transmit them as network messages, while their safety, dependability, and performance are guaranteed. We proposed a design-time static configuration of the FCP. We considered on one hand, critical control applications that are isolated and handled with a static design-time FCP configuration which reserves resources for critical applications, and on the other hand, Fog applications that are handled at runtime using pre-allocated resources. These pre-allocated resources are determined in the FCP configuration and their dimensions are optimized at design-time such that the FCP is able to accommodate upgrades in the form of future critical control applications, and dynamic changes in the form of Fog applications, minimizing their response times comparing to the typical resource reservation policies. We implemented an optimization approach using Constraint Programming, a declarative programming paradigm where realistic constraints can be added. We have evaluated our approach on several test cases and demonstrated its ability to synthesize extensible configurations.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mohammadreza Barzegaran reports financial support was provided by European Union.

Data availability

No data was used for the research described in the article.

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.

Appendix

See Table A.1.

Table A.1
Summary of the notation.

Symbol	Notation
$\mathcal{G}$	Architecture graph
$v_i \in \mathcal{V}$	Architecture node
$N_i \in \mathcal{N}$	Fog Node
$C_j \in N_i.C$	A core
$p_j \in v_i.P$	Egress port
$q_j \in p_i.Q$	Priority queue
$\epsilon_{i,j} \in \mathcal{E}$	Link
$\epsilon_{i,j}.s$	Link speed
$\epsilon_{i,j}.d$	Link propagation delay
$r_i \in \mathcal{R}$	Route
$\|r_i\|$	Number of links in a route
H	Hyperperiod
W_{p_i}	Periodic window
$W_{p_i}.c$	Window capacity
$W_{p_i}.t$	Window period
$W^j_{p_i}.\phi$	Window slice offset
$\gamma_i \in \Gamma$	Critical control application
$\tau_j \in \gamma_i.\mathcal{T}$	Critical control task
$\tau_j.t$	Critical control task period
$\tau_j.d$	Critical control task deadline
$\tau_j.c$	Critical control task WCET
$\tau^k_j.\phi$	Critical control job offset
$f_j \in \gamma_i.\mathcal{F}$	Critical control flow
$f_j.p$	Critical control flow priority
$f_j.c$	Critical control flow size
$f_j.t$	Critical control flow period
$f_j.d$	Critical control flow deadline
$\|f_i\|$	Number of critical control frames
$f^k_{i,m}.\phi$	Critical control frame offset
$\mathcal{M}$	The task mapping function to the fog nodes
$\gamma'_i \in \Gamma'$	Fog application
$\tau'_j \in \gamma'_i.\mathcal{T}$	Fog task
$\tau'_j.c$	Fog task workload
$\tau'_j.j$	Fog task arrival time
$f'_j \in \gamma'_i.\mathcal{F}$	Fog flow
$f'_j.c$	Fog flow size
$f'_j.j$	Fog flow arrival time
D_{C_i}	Defferable server
$D_{C_i}.c$	Server capacity
$D_{C_i}.t$	Server period
$D^j_{C_i}.\phi$	Server slice offset

References

[1] P. Pop, M.L. Raagaard, M. Gutierrez, W. Steiner, Enabling fog computing for industrial automation through time-sensitive networking (TSN), IEEE Commun. Stand. Mag. 2 (2) (2018) 55–61.
[2] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the Internet of Things, in: Workshop on Mobile Cloud Computing, 2012, pp. 13–16.
[3] C.-H. Chen, M.-Y. Lin, C.-C. Liu, Edge computing gateway of the industrial Internet of Things using multiple collaborative micro controllers, IEEE Netw. 32 (1) (2018) 24–32.
[4] Fog Computing and Networking Architecture Framework, IEEE 1934–2018 - IEEE standard for adoption of OpenFog reference architecture for fog computing, 2018, (Accessed 7 May 2021).
[5] A. Burns, R.I. Davis, A survey of research into mixed criticality systems, ACM Comput. Surv. 50 (6) (2018) 82.
[6] V. Karagiannis, S. Schulte, J. Leitao, N. Preguica, Enabling fog computing using self-organizing compute nodes, in: International Conference on Fog and Edge Computing, 2019, pp. 1–10.
[7] P. Pop, B. Zarrin, M. Barzegaran, S. Schulte, S. Punnekkat, J. Ruh, W. Steiner, The FORA fog computing platform for industrial IoT, Inf. Syst. 98 (2021) 101727.
[8] TTTech Computertechnik AG, Nerve, 2021, http://tttech.com/products/industrial/industrial-iot/nerve (Accessed 7 May 2021).
[9] Nebbiolo Technologies, Nebbiolo, 2021, http://www.nebbiolo.tech (Accessed 7 May 2021).
[10] Eurotech, Bridging the gap between operational technology and information technology, Red Hat, Inc. White Pap. (2016) Available: https://Bit.Ly/3rHsfnO.
[11] IEEE, Official website of the 802.1 time-sensitive networking task group, 2016, http://www.ieee802.org/1/pages/tsn.html (Accessed 7 May 2021).
[12] International Electrotechnical Commission, IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.
[13] M. Barzegaran, P. Pop, Communication scheduling for control performance in TSN-based fog computing platforms, IEEE Access 9 (2021) 50782–50797.

[14] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, O. Rana, Fog computing for the internet of things: A survey, ACM Trans. Internet Technol. 19 (2) (2019) 1–41.
[15] M. Barzegaran, A. Cervin, P. Pop, Performance optimization of control applications on fog computing platforms using scheduling and isolation, IEEE Access 8 (2020) 104085–104098.
[16] C. Avasalcai, C. Tsigkanos, S. Dustdar, Decentralized resource auctioning for latency-sensitive edge computing, in: IEEE International Conference on Edge Computing, 2019, pp. 72–76.
[17] V. Karagiannis, A. Papageorgiou, Network-integrated edge computing orchestrator for application placement, in: IEEE International Conference on Network and Service Management, 2017, pp. 1–5.
[18] OpenStack, Documentation on nova scheduler, 2020, https://docs.openstack.org/developer/nova/filter_scheduler.html (Accessed 7 May 2021).
[19] European Telecommunications Standards Institute, Mobile edge computing (MEC) framework and reference architecture (GS MEC 003 V1.1.1), 2016, http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf (Accessed 7 May 2021).
[20] O. Skarlat, V. Karagiannis, T. Rausch, K. Bachmann, S. Schulte, A framework for optimization, service placement, and runtime operation in the fog, in: IEEE International Conference on Utility and Cloud Computing, 2018, pp. 164–173.
[21] L. Almeida, Response time analysis and server design for hierarchical scheduling, in: Proceedings of the IEEE Real-Time Systems Symposium Work-in-Progress, Citeseer, 2003.
[22] V. Gavrilut, B. Zarrin, P. Pop, S. Samii, Fault-tolerant topology and routing synthesis for IEEE time-sensitive networking, in: Proceedings of the 25th International Conference on Real-Time Networks and Systems, 2017, pp. 267–276.
[23] IEEE standard for local and metropolitan area network–bridges and bridged networks, in: IEEE Std 802.1Q-2018, Revision of IEEE Std 802.1Q-2014, 2018, pp. 1–1993.
[24] IEEE standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications, in: IEEE Std 802.1AS-2020, Revision of IEEE Std 802.1AS-2011, 2020, pp. 1–421.
[25] S.S. Craciunas, R.S. Oliver, M. Chmelík, W. Steiner, Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks, in: Proceedings of the International Conference on Real-Time Networks and Systems, 2016, pp. 183–192.
[26] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, Springer Science & Business Media, 2011.
[27] M. Verucchi, M. Theile, M. Caccamo, M. Bertogna, Latency-aware generation of single-rate DAGs from multi-rate task sets, in: 2020 IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS, 2020, pp. 226–238, http://dx.doi.org/10.1109/RTAS48715.2020.000-4.
[28] G.C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Springer, 2011.
[29] I. Shin, I. Lee, Compositional real-time scheduling framework with periodic model, ACM Trans. Embed. Comput. Syst. 7 (3) (2008) 1–39.
[30] I. Shin, I. Lee, Periodic resource model for compositional real-time guarantees, in: IEEE Real-Time Systems Symposium, 2003, pp. 2–13.
[31] J.K. Strosnider, J.P. Lehoczky, L. Sha, The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments, IEEE Trans. Comput. 44 (1) (1995) 73–91.
[32] O. Sinnen, Task Scheduling for Parallel Systems, Vol. 60, John Wiley & Sons, 2007.
[33] J.D. Ullman, NP-complete scheduling problems, J. Comput. System Sci. 10 (3) (1975) 384–393.
[34] A. Cervin, P. Pazzaglia, M. Barzegaran, R. Mahfouzi, Using jittertime to analyze transient performance in adaptive and reconfigurable control systems, in: IEEE International Conference on Emerging Technologies and Factory Automation, 2019, pp. 1025–1032.
[35] W. Zheng, J. Chong, C. Pinello, S. Kanajan, A. Sangiovanni-Vincentelli, Extensible and scalable time triggered scheduling, in: International Conference on Application of Concurrency to System Design, 2005, pp. 132–141.
[36] Q. Zhu, Y. Yang, E. Scholte, M.D. Natale, A. Sangiovanni-Vincentelli, Optimizing extensibility in hard real-time distributed systems, in: Real-Time and Embedded Technology and Applications Symposium, 2009, pp. 275–284.
[37] G. Lipari, E. Bini, Resource partitioning among real-time applications, in: Proceedings of the Euromicro Conference on Real-Time Systems, 2003, pp. 151–158.
[38] G. Buttazzo, G. Lipari, L. Abeni, M. Caccamo, Soft Real-Time Systems, Springer, 2005.
[39] N. Audsley, A. Burns, M. Richardson, K. Tindell, A.J. Wellings, Applying new scheduling theory to static priority pre-emptive scheduling, Softw. Eng. J. 8 (5) (1993) 284–292.
[40] X. Feng, A.K. Mok, A model of hierarchical real-time virtual resources, in: Real-Time Systems Symposium, 2002, pp. 26–35.
[41] M. Barzegaran, V. Karagiannis, C. Avasalcai, P. Pop, S. Schulte, S. Dustdar, Towards extensibility-aware scheduling of industrial applications on fog nodes, in: International Conference on Edge Computing, 2020, pp. 67–75.
[42] Google, Google OR-tools, 2021, https://developers.google.com/optimization (Accessed 7 May 2021).
[43] M. Barzegaran, B. Zarrin, P. Pop, Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming, in: 2nd Workshop on Fog Computing and the IoT, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020, pp. 1–4.
[44] E.K. Burke, G. Kendall, et al., Search Methodologies, Springer, 2005.
[45] M. Barzegaran, N. Desai, J. Qian, K. Tange, B. Zarrin, P. Pop, J. Kuusela, Fogification of electric drives: An industrial use case, in: IEEE International Conference on Emerging Technologies and Factory Automation, 2020, pp. 1–5.
[46] M. Barzegaran, N. Desai, J. Qian, P. Pop, Electric drives as fog nodes in a fog computing-based industrial use case, J. Eng. (2021).
[47] B. Lincoln, A. Cervin, Jitterbug: A tool for analysis of real-time control performance, in: Proceedings of the IEEE Conference on Decision and Control, Vol. 2, 2002, pp. 1319–1324.
[48] Y. Wang, Y. Zhang, Y. Su, X. Wang, X. Chen, W. Ji, F. Shi, An adaptive and hierarchical task scheduling scheme for multi-core clusters, Parallel Comput. 40 (10) (2014) 611–627.
[49] K. Lakshmanan, R. Rajkumar, J. Lehoczky, Partitioned fixed-priority preemptive scheduling for multi-core processors, in: Euromicro Conference on Real-Time Systems, IEEE, 2009, pp. 239–248.
[50] L. Yin, J. Luo, H. Luo, Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing, IEEE Trans. Ind. Inf. 14 (10) (2018) 4712–4721.
[51] K.M. Zuberi, K.G. Shin, Design and implementation of efficient message scheduling for controller area network, IEEE Trans. Comput. 49 (2) (2000) 182–188.
[52] M. Gagliardi, R. Rajkumar, L. Sha, Designing for evolvability: Building blocks for evolvable real-time systems, in: Proceedings Real-Time Technology and Applications, IEEE, 1996, pp. 100–109.
[53] D. Rowe, J. Leaney, D. Lowe, Defining systems evolvability-a taxonomy of change, Change 94 (1994) 541–545.
[54] J. Regehr, A. Reid, K. Webb, M. Parker, J. Lepreau, Evolving real-time systems using hierarchical scheduling and concurrency analysis, in: IEEE Real-Time Systems Symposium, 2003, pp. 25–36.
[55] A. Kornecki, J. Zalewski, Certification of software for real-time safety-critical systems: state of the art, Innov. Syst. Softw. Eng. 5 (2) (2009) 149–161.
[56] P. Pop, P. Eles, Z. Peng, T. Pop, Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 12 (8) (2004) 793–811.
[57] Y. Wang, F. Huang, Y. Li, B. Pan, Y. Wu, Hierarchical scheduling and real-time analysis for vehicular time-sensitive network, in: International Symposium on Computational Intelligence and Design, Vol. 2, 2019, pp. 23–26.
[58] L. Guo, A. Ghosal, H. Zeng, P. Giusto, A. Sangiovanni-Vincentelli, Methods and tools for calculating the flexibility of automotive HW/SW architectures, SAE Int. J. Passenger Cars-Electronic Electr. Syst. 5 (2012-01-0005) (2012) 17–26.
[59] A. Mehiaoui, E. Wozniak, S. Tucci-Piergiovanni, C. Mraidha, M. Di Natale, H. Zeng, J.-P. Babau, L. Lemarchand, S. Gerard, A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems, in: Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems, 2013, pp. 121–132.
[60] Q. Zhu, H. Zeng, W. Zheng, M.D. Natale, A. Sangiovanni-Vincentelli, Optimization of task allocation and priority assignment in hard real-time distributed systems, ACM Trans. Embed. Comput. Syst. 11 (4) (2013) 1–30.