IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 11, 2020, accepted May 26, 2020, date of publication June 1, 2020, date of current version June 15, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2999322*

# Performance Optimization of Control Applications on Fog Computing Platforms Using Scheduling and Isolation

**MOHAMMADREZA BARZEGARAN[1], ANTON CERVIN[2], (Member, IEEE), AND PAUL POP[1], (Member, IEEE)**

[1]Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Kongens Lyngby, Denmark
[2]Department of Automatic Control, Lund University, 221 00 Lund, Sweden

Corresponding author: Mohammadreza Barzegaran (mohba@dtu.dk)

This work was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant agreement No. 764785 for the FORA (Fog Computing for Robotics and Industrial Automation) project.

**ABSTRACT** In this paper, we address mixed-criticality applications characterized by their safety criticality and time-dependent performance, which are virtualized on a Fog Computing Platform (*FCP*). The *FCP* is implemented as a set of interconnected multicore computing nodes, and brings computation and communication closer to the edge of the network, where the machines are located in industrial applications. We use partitioning and static-cyclic scheduling to provide isolation among mixed-criticality tasks and to guarantee their timing requirements. The temporal and spatial isolation is enforced via partitions, which execute tasks with the same criticality level. We consider that the tasks are scheduled using static cyclic scheduling. We are interested in determining the mapping of tasks to the cores of the fog nodes, the assignment of tasks to the partitions, the partition schedule tables, and the tasks' schedule tables, such that the Quality-of-Control for the control tasks is maximized and we meet the timing requirements for all tasks, including tasks with lower-criticality levels. We are also interested in determining the periods for control tasks to balance the schedulability and the control performance. We have proposed a Simulated Annealing metaheuristic, which relies on a heuristic algorithm for determining the schedules and partitions, to solve this optimization problem. Our optimization strategy has been evaluated on several test cases, showing the effectiveness of the proposed method.

**INDEX TERMS** Fog computing, mixed-criticality systems, quality-of-control, scheduling, partitioning, optimization.

## I. INTRODUCTION

We are at the beginning of a new industrial revolution, i.e., Industry 4.0, which is underpinned by a digital transformation that will affect all industries. Industry 4.0 will bring increased productivity and flexibility, mass customization, reduced time-to-market, improved product quality, innovations and new business models. However, Industry 4.0 will only become a reality through the convergence of Operational and Information Technologies (*OT* & *IT*), which use different computation and communication technologies. *OT* consists of cyber-physical systems that monitor and control physical processes that manage, e.g., automated manufacturing, critical infrastructures, smart buildings and smart cities. These application areas are typically safety critical and real-time, requiring guaranteed extra-functional properties, such as, real-time behavior, reliability, availability, industry-specific safety standards, and security.

*OT* uses proprietary solutions imposing severe restrictions on the information flow. *IT* such as Cloud Computing cannot be used at the edge of the network, where industrial machines are located, and where very stringent extra-functional properties have to be guaranteed [1]. Instead, a new paradigm, called *Fog Computing*, is envisioned as an architectural means to realize the *IT*/*OT* convergence. Fog Computing is a "system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things" [2]. With Fog

The associate editor coordinating the review of this manuscript and approving it for publication was Huaqing Li.

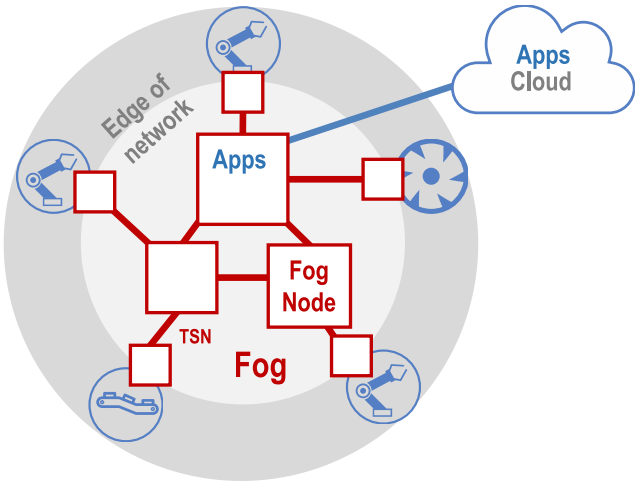



**FIGURE 1.** Fog Computing platform. Boxes represent fog nodes, connected with each other and to the Cloud; the thick lines are the network. Applications (Apps) run in the fog and Cloud.

Computing, communication devices, such as switches and routers are extended with computational and storage resources to enable a variety of communication and computation options (see Fig. 1).

Fog Computing will enable a powerful convergence, unification and standardization at the networking, security, data, computing, and control levels. It will lead to improved interoperability, security, more efficient and rich control, and higher manufacturing efficiency and flexibility [3]. The vision is to virtualize the control (which is implemented as control tasks running on a Fog Computing Platform) and achieve the same level dependability as the one taken for granted in *OT*. Several initiatives are currently working towards realizing this vision [4], [5].

The integration of computational and storage resources into the communication devices is realized in the *fog node* (*FN*). In many applications, including industrial automation and robotics, several layers of *FN*s with differing computation, communication and storage capabilities will evolve, from powerful high-end *FN*s to low-end *FN*s with limited resources. Researches have started to propose solutions for the implementation of *FN*s [3], [4] and fog node solutions have started to be developed by companies [4]–[6].

An *FN* is equipped with computational resources that allows the execution of applications and it is connected to a larger data processing facility like a Cloud environment. Regarding computation, we assume that the control tasks are running in an Real-Time Operating System using real-time scheduling policies (we consider static-cyclic scheduling in this paper), and the control applications are separated in different *partitions* enforced using hardware-supported virtualization, based on hypervisors, such as ACRN [7] or PikeOS [8]. *FN*s could be connected to each others and to the machines through a deterministic communication solution, such as IEEE 802.1 Time-Sensitive Networking (TSN) [9], see Fig. 1. Such a Fog Computing Platform (*FCP*) allows to increase the spatial distance between the physical process and the *FN* that controls it, allowing the control functions can be executed remotely on the *FN*. However, the way the *FCP* is configured has an impact on the control performance of the control applications.

Given a set of mixed-criticality applications and an *FCP*, we are interested to determine an *FCP* configuration such that the Quality-of-Control (*QoC*) of control tasks is maximized and all the tasks meet their deadlines. Determining an *FCP* configuration means deciding on the partitions, the mapping of tasks to the *FN*s and partitions, the schedule table for tasks, the partition table for partitions, and the periods of control tasks. We do not address the scheduling of messages on the TSN network, which can be solved with approaches such as [10] that achieve low latency and zero jitter.

### A. CONTRIBUTIONS

This paper shows that when control becomes virtualized, implemented as tasks on an *FCP*, the configuration of the *FCP* has a strong impact on the control performance. We formulate the *FCP* configuration as an optimization problem, and we have proposed a metaheuristic solution to solve it. Compared to the related work (see Sect. VII), which has addressed the scheduling of tasks to maximize *QoC*, we also optimize the partitioning, which is required in an *FCP* to provide isolation among mixed-criticality applications, decide on the mapping of tasks to partitions, consider the preemption of tasks to make static schedules more flexible, and determine the period of control tasks to trade-off *QoC* and schedulability of non-control tasks. In addition, we also consider a more realistic model of control applications and provide more accurate measure of *QoC* compared to previous work, see Sect. IV.

### B. OUTLINE OF THE PAPER

In the remainder of this paper, we give the models for application and architecture of the system in Sect. II. The problem is formulated in Sect. III. We give an introduction to control theory in Sect. IV. In Sect. V, we present the details of our proposed optimization strategy with an illustrative example. Our optimization approach is evaluated in Sect. VI on several test cases. The related work is covered in Sect. VII and Sect. VIII concludes the paper.

## II. SYSTEM MODEL

This section presents the architecture and application models. Table.1 summarizes the notations used in the system model.

### A. ARCHITECTURE MODEL

There have been several *FN* architectures proposed, and some of them are commercially realized [4]–[6], [11], [12]. A possible *FN* architecture targeting mixed-criticality applications, is presented in Fig. 2. Such an architecture is similar to several *FN* architectures prepared for industrial applications [4], [6]. We model the architecture as a set of *FN*s, denoted by $\mathcal{N}$. Each *FN*, $N_i \in \mathcal{N}$, has a set of cores $\mathcal{P}_i$, and each core is denoted with $P_j \in \mathcal{P}_i$. An example architecture with two *FN*s

**TABLE 1. Summary of notations.**

| Symbol | Fog Computing Platform (FCP) |
|---|---|
| $\mathcal{N}$ | Set of all Fog Nodes |
| $N_i(\mathcal{P}_i) \in \mathcal{N}$ | Fog Node (*FN*) |
| $\mathcal{P}_i$ | Set of all cores in the Fog Node $N_i$ |
| $P_j \in \mathcal{P}_i$ | Core |
| $\mathfrak{P}$ | Set of all cores in the platform |
| $\Delta$ | Set of all partitions |
| $\delta_i(L_i, \xi_i) \in \Delta$ | Partition |
| $L_i$ | Criticality level of a partition |
| $\xi_i$ | Overhead time of a partition |
| $\mathcal{V}$ | Set of partition tables |
| $v_i \in \mathcal{V}$ | Partition table |
| $\mathcal{S}$ | Set of schedule tables |
| $s_i \in \mathcal{S}$ | Schedule table |
| $\Gamma$ | Set of all applications |
| $\gamma_i \in \Gamma$ | Set of tasks of application $\gamma_i$ |
| $Fi$ | Set of possible periods for an application |
| $\mathcal{F}$ | The application mapping function to periods |
| $\tau_i(D_i, T_i, L_i, \mathcal{C}_i) \in \gamma_j$ | Task |
| $D_i$ | Deadline of a task |
| $T_i$ | Period of a task |
| $L_i$ | Criticality level of the application $\gamma_j$ |
| $\mathcal{C}_i$ | Set of WCETs for the application $\gamma_j$ |
| $\mathcal{M}$ | The task mapping function to the cores |
| $\mathcal{O}$ | The task assignment function to the partitions |

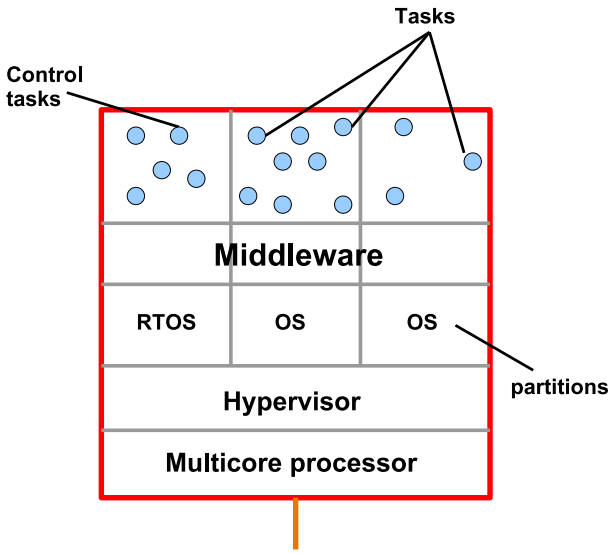



**FIGURE 2. Fog node architecture. The software stack is running on a multicore, and has, from bottom to top, a hypervisor, partitions running *OS*es, middleware and tasks.**

is presented in Fig. 3. The *FN*s have respectively two and one cores. Sensors and actuators are connected to *FN*s with network switches. The lines represent network links.

Mixed-criticality applications sharing the same platform have to be isolated from each other, otherwise a faulty lower-criticality task may interfere with a higher-criticality task, leading to failure. We assume that the applications are isolated from each other using spatial and temporal partitioning [13], implemented via hypervisors such as ACRN [7], Xen [14], PikeOS [8] or XtratuM [15].

We denote the set of partitions with $\Delta$. Each partition $\delta_i \in \Delta$, is characterized by a criticality level $L_i$. For example

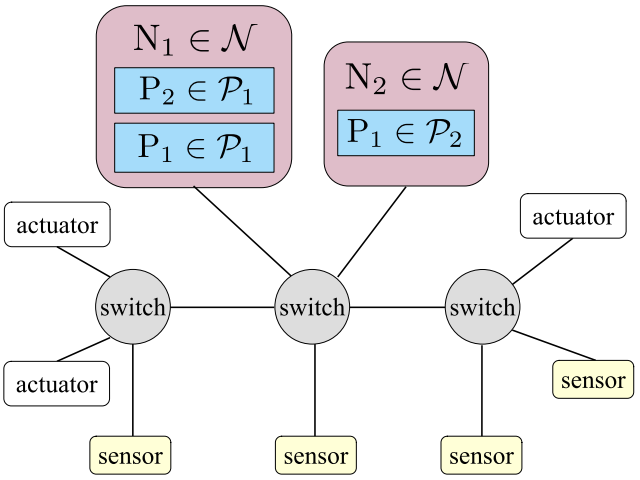



**FIGURE 3. Example architecture with two *FN*s.**

$L_i$ can represent the Safety Integrity Levels (SIL) of an application, which has values from 0, non-critical, to 4, highest criticality [16]. A partition $\delta_i$, scheduled on multiple cores, consists of several partition slices that are time slots to which the processor is assigned for the partition. We assume that the partitions are statically scheduled via partition tables, denoted with $\mathcal{V}$, (e.g, as used in Xen or PikeOS), which allocate processing cores from an *FN* to partitions in partition slices. A partition table repeats periodically with a system cycle. Switching among partition slices imposes an overhead. This overhead depends on the computing platform, the hypervisor, see [17] for a description of overheads in Xen, and may also depend on the contents of the partition. For example in [18], researchers assume that the overhead is 5% of the maximum worst-case execution time (WCET) of the tasks allocated to a partition. Our model is general, and assumes a partition-dependent overhead denoted with $\xi_i$ for each partition $\delta_i$.

Real-time applications can be implemented with time-triggered or event-triggered scheduling policies. In this paper, we assume that the scheduling policy is static cyclic scheduling [19] (also known as time-triggered scheduling), which has been shown to be suitable for critical control applications. We will consider event-triggered scheduling in our future work. The set of all schedule tables in the model are denoted with $\mathcal{S}$. A schedule table $s_i \in \mathcal{S}$, captures the start and finishing time of tasks. We consider that within a schedule table, a task may be split into several parts, similar to run-time preemption in preemptive scheduling, but decided at design time. This has been shown to improve flexibility, schedulability [20] and *QoC* for control tasks [21]. The preemption threshold can be controlled by a parameter called macrotick, which specifies the granularity of preemption [22].

Fig. 4 shows an example of partition tables $\mathcal{V}$ and schedule tables $\mathcal{S}$ using a Gantt chart. In this example, we assume that mixed-criticality applications with a total of ten tasks are executing on two cores, which have four partitions $\Delta$. All tasks in a partition have the same criticality level. The partitions $\delta_1$, $\delta_2$, $\delta_3$, and $\delta_4$ have the criticality level of

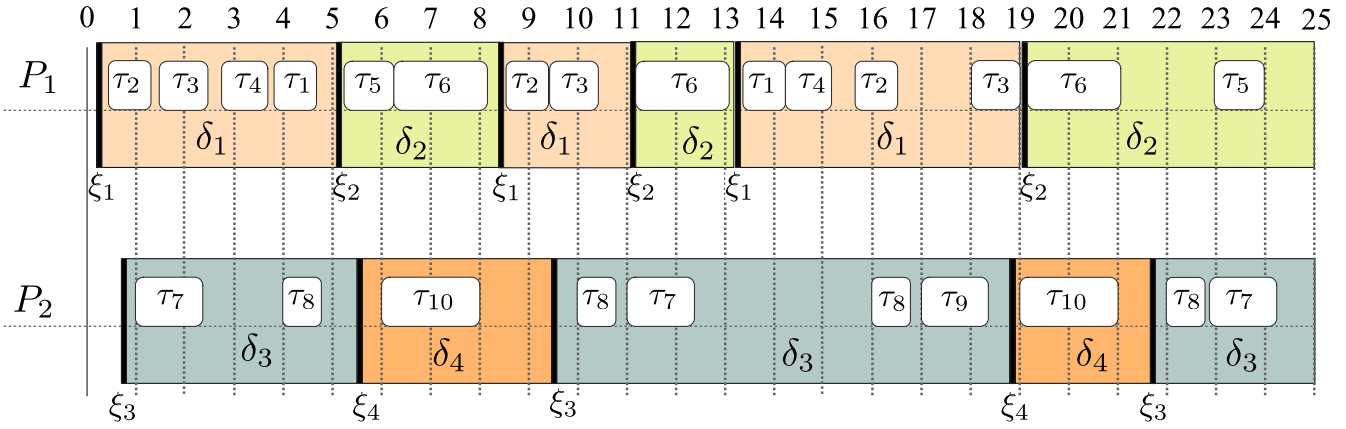



**FIGURE 4.** Example partition tables and schedule tables: Ten tasks are scheduled on two cores and mapped to four partitions.

respectively $L_1 = 1$, $L_2 = 0$, $L_3 = 3$, and $L_4 = 2$. The black lines represent the overhead times of the partitions. The task scheduling is depicted with white rectangles and the partition scheduling is depicted in colored rectangles.

### B. APPLICATION MODEL

The set of all applications is denoted with $\Gamma$. An application is denoted with $\gamma_i \in \Gamma$ and composed of tasks $\tau_j \in \gamma_i$. Tasks may have data dependencies, which are modeled using a directed acyclic graph (DAG), where nodes are tasks and edges represent data flows between the tasks. A data-dependent task is ready when all of its inputs have arrived. A task produces its outputs when it terminates. For example, as will be discussed in Sect.IV-A, each control application is implemented as three data-dependent control tasks: a sampling task, a task that implements the control algorithm and an actuator task. Each task $\tau_i$ is periodic and has a period $T_i$, and a deadline $D_i$. The deadline is relative to the activation of the task. For each task $\tau_i$, we know the set of worst-case execution times (WCETs) $\mathcal{C}_i$ on the cores, where it is considered for mapping. The WCETs may be impacted by shared resources in a multicore, i.e., bus, memory, I/O. However, the problem of contention-aware scheduling is orthogonal to our work and we can use the techniques mentioned in [23] to account for the contention.

The mapping of tasks to the cores is modelled by using the function $\mathcal{M} : \tau_i \longrightarrow \mathfrak{P}$, where $\mathfrak{P}$ is the set of all cores in the platform. The system engineers may place constraints on the mapping of tasks, which can be handled by our model. The tasks are also assigned to partitions for execution. The assignment of the tasks to the the partitions is denoted by $\mathcal{O} : \tau_i \rightarrow \Delta$, where $\Delta$ is the set of all partitions in the system. The criticality level of an application $\gamma_i$ is captured by its SIL $L_i$, see Sect. II-A. Tasks can be assigned only to partitions that have the same criticality level.

We assume that tasks which have data dependencies share the same period. For a control application $\gamma_i$, we are given a set of possible periods $F_j$. We use the function $\mathcal{F}$ to capture the period $T_i = \mathcal{F}(\gamma_i) \in F_i$ of a control application $\gamma_i$. Our optimization strategy will select the period of the control application.

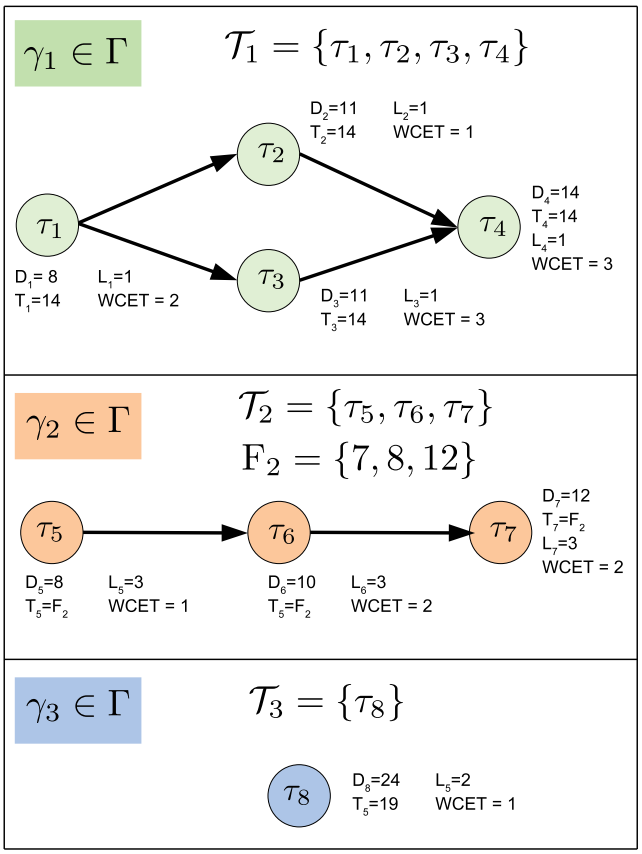



**FIGURE 5.** Example application model with three applications.

We show in Fig. 5 an example application model consisting of three applications. The criticality level, deadline and period of each task are depicted in the figure. The WCET of each task is also given considering a given mapping to a core. The values for deadlines, periods, and WCETs are in milliseconds. The application $\gamma_2$ is a control application with

three tasks with precedence constraints. The application $\gamma_2$ has a set of possible periods $F_2$.

## III. PROBLEM FORMULATION

We formulate the problem as follows: Given (i) a set of applications $\Gamma$ and (ii) a set of *FN*s $\mathcal{N}$, we want to determine a configuration $\Psi$ consisting of: (1) a set of partitions $\Delta$, (2) a mapping $\mathcal{M}$ of the tasks to cores, (3) an assignment $\mathcal{O}$ of tasks to the partitions, (4) the periods of control applications $\mathcal{F}$, (5) the partition tables $\mathcal{V}$, and (6) the schedule tables $\mathcal{S}$ such that:

1) **Maximum control performance is achieved for the critical control applications**: We seek a solution which has the best overall *QoC* for all the control applications. This is realized by minimizing the function $\bar{J}$ captured by Eq. (7), see Sect. V-C.
2) **The deviation among the *QoC* of control applications is minimized**: We would like to balance the deviation $\sigma_J$, captured by Eq. (8), see Sect. V-C.
3) **Temporal isolation is achieved among tasks with different criticality levels**: Each task $\tau_i$ and its assigned partition $\delta_j$, captured with the function $\mathcal{O}$, share the same criticality level.
4) **The deadlines for all tasks are met**: Given that all the tasks are periodic and real-time, each task $\tau_i$ should completed before its deadline $D_i$.

## IV. CONTROL THEORY

The mathematical relation between the inputs, outputs and state variables of a dynamical system around an equilibrium point can be modelled as a linear differential equation and denoted by a state-space representation [24]

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + w(t), \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{1}$$

where the vectors $x$, $u$, $w$ and $y$ denote the state, the control input, the disturbance input, and the measured output respectively, and where $A$, $B$, $C$ and $D$ are matrices of appropriate sizes. The input–output relationship can equivalently be described by a transfer function $G(s)$ [24]. A feedback control system, *FCS*, or simply a control application, samples the output of the dynamical system $y(t)$, calculates the deviation from the desired output $r(t)$ (in this paper generically assumed to be zero), and drives the deviation to zero by applying an appropriate control signal $u(t)$.

### A. FEEDBACK CONTROL SYSTEM

An *FCS* can be implemented as a three task application. The source task, let's call it $\tau_1$, samples the dynamical system by using sensors. The task may process the captured data from sensors. The second task, let's call it $\tau_2$, uses the output of the task $\tau_1$ to calculate the control signal. The task $\tau_2$ utilizes various methods for the calculation and may be engaged with time-consuming calculation [24]. The implemented method for calculating the control signal is called the control law. The sink task, let's call it $\tau_3$, uses the output of the task $\tau_2$ to exert the control signal using the connection with the actuators. A simple *FCS* has an analogue to digital converter for the source task $\tau_1$, a control law $\tau_2$, and an analogue to digital converter for the sink task $\tau_3$. Fig. 6 shows a simple *FCS*.

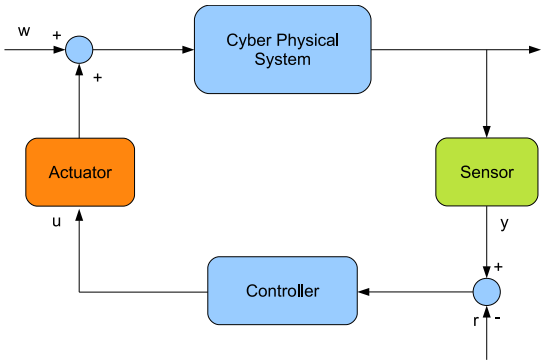



**FIGURE 6.** A simple *FCS*.

A control application is typically a periodic application with a known period. The period should be chosen in relation to the speed of the controlled system, and the shorter the period, the faster the controller is able to respond to the typical disturbances. On the other hand, a too short period causes high utilization of resources and leads to problems in resource-constrained computing platforms. A common rule of thumb [25] is to determine the period of the application based on the bandwidth of the closed-loop system. The closed-loop transfer function $H(s)$ is calculated by

$$H(s) = \frac{G(s)K(s)}{1 + G(s)K(s)}, \tag{2}$$

where $G(s)$ and $K(s)$ are the transfer functions of the dynamical system and the feedback controller respectively [24]. The sampling period $T$ is then chosen in the interval

$$\frac{0.2}{\omega_b} \leq T \leq \frac{0.6}{\omega_b}, \tag{3}$$

where $\omega_b$ is the 3 dB bandwidth of $H(s)$ [25].

As discussed, choosing the period from the interval has two impacts; first, stability and robustness of controller and last, resource utilization and schedulability. Our optimization strategy will determine the periods $\mathcal{F}$ to strike a compromise between their impacts.

The task timing is a source of additional disturbances for a control application. Ideally, the controller should execute without timing variations (jitter) and with as short delay as possible between the sensor task and the actuator task. A time delay has the direct consequence of decreasing the phase margin of the control system, which means worse performance and less robustness. Jitter is the deviation from the true periodic timing of an event, and its effects on the control performance are less obvious to analyze. In a control application, the event can be the execution of a task or the receiving of a network message.

The execution of a task is a periodic event of which instances are characterized by start time, duration and end

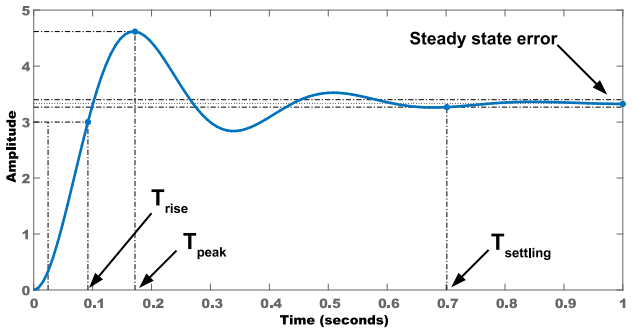



**FIGURE 7.** Step response of a sample control loop.

time. Jitters can be associated with the start time, the end time and the duration. It is also associated either among all instances or two consecutive instances. The data packets to/from actuators/sensors are also periodic events which are characterized by send-time, transmit-time and receive-time which are vulnerable to jitter. While we are ignoring the communication in this paper, delays and jitter are only applied to tasks. The jitter of a task is either measured among all the instances (absolute) or two consecutive instances (relative). We categorize jitters as follows:

- **Start Jitter** of a task is the maximum deviation of the arrivals of instances of a task.
- **Release Jitter** of a task is the maximum deviation of the worst-case delay between the arrivals of instances of a task and their release times.
- **End Jitter** of a task is the maximum deviation of the release time of instances of a task.
- **Input–Output Jitter** is the maximum deviation of the worst-case delay between sampling from a cyber-physical system and exerting the actuation to it among the instances of tasks in a control loop. This type of jitter covers both timing of communication links form and to sensors and actuators, and the execution of the control tasks.

### B. CONTROL DESIGN

While designing an *FCS*, there is a trade-off between accuracy and rapidity of the control loop. The trade-off is called the control performance. It is determined by several parameters such as the damping ratio, the phase margin and the gain margin, see [24] for more details. These parameters help control engineers to find the suitable control law and tune the control law to get the intended performance. The accuracy and rapidity is depicted in the transient and steady state response of the control loop. Fig. 7 shows the transient and steady state step-response of a sample control loop with notation of associated parameters (rise-time $T_{\text{rise}}$, peak-time $T_{\text{peak}}$, settling-time $T_{\text{settling}}$ and steady state error).

The rise-time $T_{rise}$ is defined as the time takes for the output response to reach 90% of the input value. The rise-time shows how fast the controller can react to the disturbances exerted to the dynamical system. The peak response is defined as highest out-put response the controller reached before the desired value. The peak plays an important role in the robustness of the controller against disturbances. The settling-time $T_{settling}$ is defined as the time takes for the output response to reach 98% of the input value. The settling-time shows how fast the controller can reach to the desired state. The steady-state error shows the minimum deviation of the controller output response from the desired state. It shows the accuracy of the controller.

The various aspects of the control loop performance can be captured in a cost function. A common choice [26] is to use a quadratic cost function of the form

$$J = \int_0^\infty \left( x^T(t) Q_1 x(t) + u^T(t) Q_2 u(t) \right) dt, \tag{4}$$

where the weighting matrices $Q_1$ and $Q_2$ tell how much deviations in the different states and the control input should be penalized. By proper tuning of the cost function, the desired transient and steady-state behaviour can be achieved in the control design. The same cost function can also be used to evaluate the performance of the controller under non-ideal circumstances. A larger value of the cost $J$ then means that the response is more sluggish or more oscillatory, typically increasing the settling time of the system.

Given a linear system description by Eq. (1) and a quadratic cost function in Eq. (4), an optimal controller known as a linear-quadratic-Gaussian (LQG) controller can be calculated [24]. The above formulation is given in continuous time, but the LQG design methodology can also handle a large number of other conditions, such as sampled design [25] and compensation for time delays [27]. The Jitterbug toolbox [28], utilized in this paper to design control applications, has support for designing an optimal sampled LQG controller that compensates for either a fixed or a random input–output delay with a given probability distribution.

### C. CALCULATION OF CONTROL PERFORMANCE

In this paper, we use JITTERTIME [29] to calculate the *QoC* with the cost function $J$, defined in Eq. (4). JITTERTIME takes the schedule tables $\mathcal{S}$ and partition tables $\mathcal{V}$ and calculates the cost $J$. The tables $\mathcal{S}$ contain the starting and finishing time of the tasks. JITTERTIME simulates the behaviour of a control application with the given starting and finishing times of control tasks and evaluates the behaviour using the quadratic cost function in Eq. (4).

The cost $J$ decreases under the circumstances in which the Input-Output Jitter of a control application (defined in Sect. IV-A) as well as the end-to-end response of the control application decreases. The end-to-end response of a control application is the delay between the sampling from a cyber-physical system and exerting the actuation to it. In our problem, the delay is between the starting of the sensor task to the finishing of the actuator task. More information about the inner workings of JITTERTIME can be found in [29].

## V. SOLUTION

The problem we are addressing in this paper is interactable. Finding a solution to our problem involves deciding on the schedule tables, which has been shown to be Non-deterministic Polynomial time (NP)–complete in the strong sense [30]. For such problems, exact optimization methods such as Brunch & Bound, Integer Linear Programming and Constraint Programming have exponential efforts. Hence, we propose a Simulated Annealing (*SA*)-based metaheuristic [31] to solve this optimization problem. Metaheuristics do not guarantee finding the optimal solution, but have been shown to find good quality solutions for a wide range of practical applications [31].

We have decided to divide the problem such that the schedule synthesis is performed separately within the *SA* using a scheduling heuristic. List Scheduling [32] is a typical heuristic that derives good quality solutions, but it cannot easily handle applications with multiple periods and preemption. Instead, inspired by [22], we have proposed a scheduling heuristic based on the simulation of an Earliest Deadline First (*EDF*) algorithm, which can handle both multiple periods and preemption.

An overview of our proposed Fog Computing Platform Configuration (FCPC) optimization strategy in shown in Alg.1. The *SA* decides the period of the control applications, the mapping of tasks to cores in the *FCP*. The assignment of tasks to partitions and the partition and schedule tables are decided by our *EDF*-based Scheduling and Partitioning Heuristics (SPH, called in Alg.1). *SA* also decides parameters that influence the scheduling in SPH, such as task offsets $\Theta$ and relative deadlines $\Phi$ used for the *EDF* simulation.

*SA* is presented in Sect. V-A and SPH in Sect. V-B. The objective function used for the optimization is presented in Sect. V-C. *SA* uses design transformation to explore the search space, and these are presented in Sect. V-D. Sect. V-E has an example that illustrates how our proposed FCPC strategy works.

### A. SIMULATED ANNEALING

*SA* (line 7–17 in Alg. 1) starts from an initial solution (line 4) and iterates to search the solution space (line 7–17). The initial solution assigns the period of each control task to the minimum value in its set of periods $\mathcal{F}$, assigns the offsets $\Theta$ of tasks to zero, and sets the relative deadline $\Phi$ of all tasks to their deadline values. The initial mapping $\mathcal{M}$ is obtained by a greedy approach i.e., each task is mapped iteratively to the core that has the smaller utilization in that iteration. The initial assignment of tasks to partitions $\mathcal{O}$ is defined such that each application has a partition for its criticality level on each core where the application has a task mapped. The partition tables $\mathcal{V}$ and schedule tables $\mathcal{S}$ are obtained with our Scheduling and Partitioning Heuristic (SPH), called inside the InitialSolution function.

In each iteration, *SA* uses design transformations (or *moves*) to generate neighboring solutions starting from the current solution $\Psi$ (line 8). The generated neighborhood is evaluated with the cost function $\Omega$, defined in Sect. V-C. In each iteration, the algorithm compares the cost $\Omega_i$ of the generated neighborhood with the cost $\Omega$ of the current solution (line 13).

*SA* accepts a solution if the cost is improved. *SA* may also accept a worse-quality solution (in the hope to better explore the solution space) with a certain probability:

$$Prob(\lambda, t) = e^{-\frac{\lambda}{t}}, \tag{5}$$

where $\lambda$ is the difference between cost of the generated neighborhood and cost of the current solution (line 12). The probability to accept worse solutions decreases with time according to a ''cooling schedule'', where $t$ is the current temperature. *SA* starts from an initial temperature $T_{start}$ (line 2), and cools down in each iteration at the rate of $\alpha$ (line 16). The search terminates when a stopping criterion has been satisfied (line 17), e.g., no improvement after a given number of iterations, a temperature of zero or a time limit was reached.

**Algorithm 1** $\Psi =< \mathcal{M}, \mathcal{O}, \mathcal{S}, \mathcal{V}, \mathcal{F} >= FCPC(\Gamma, \mathcal{N})$

```
1:  i ← 0
2:  t ← T_start
3:  Θ ← {0}; Φ ← {D_i}
4:  Ψ ← InitialSolution(Γ, N)
5:  J ← JitterTime(S, V, Γ)
6:  Ω ← CostFunction(J, S)
7:  repeat
8:      < M_i, , F_i, Θ_i, Φ_i > ← Neighbor(Ψ, Γ, N)
9:      < S_i, V_i > ← SPH(M_i, F_i, Θ_i, Φ_i, Γ, N)
10:     J_i ← JitterTime(S_i, V_i, Γ)
11:     Ω_i ← CostFunction(J_i, S_i)
12:     λ ← Ω_i − Ω
13:     if λ < 0 or random[0, 1) < Prob(λ, t) then
14:         Ψ ← Ψ_i; Θ ← Θ_i; Φ ← Φ_i
15:     end if
16:     t ← t × α
17: until stopping criterion is True
18: return Ψ =< M, O, S, V, F >
```

### B. SCHEDULING AND PARTITIONING HEURISTIC (SPH)

Our proposed Scheduling and Partitioning Heuristic (SPH) is presented in Alg.2 and takes as input a mapping $\mathcal{M}$, a set of periods $\mathcal{F}$, a set offsets $\Theta$, a set of relative-deadlines $\Phi$, the set of applications $\Gamma$, and the set of *FN*s $\mathcal{N}$. The main idea of SPH is to create first a schedule table for the tasks considering the mapping fixed by SA, and then to post-process the schedule table to derive the partitions and the allocation of tasks to the partition slices. Thus, SPH has two parts, the first part schedules the tasks (line 2–10) and the second part (line 11–13) groups the tasks together to form partitions. During the construction of the schedule table in the first phase, the SPH does not consider the partitioning.

As mentioned earlier, to derive the schedule tables $\mathcal{S}$, we perform at design time an *EDF* simulation. The output of that

**Algorithm 2** $< \mathcal{O}, \mathcal{S}, \mathcal{V} >= SPH(\mathcal{M}, \mathcal{F}, \Theta, \Phi, \Gamma, \mathcal{N})$

1: $H \leftarrow$ HyperPeriod($\Gamma$)
2: $Q_{jobs} \leftarrow$ CreateJobs($\Gamma, H$)
3: $t \leftarrow 0$
4: **repeat**
5: **for** all $\partial$ in $Q_{jobs}$ ready at $t$ **do**
6: $\partial_H \leftarrow$ GetHighestPriority($\partial$)
7: $\mathcal{X} \leftarrow$ Schedule($\partial_H$)
8: **end for**
9: $t \leftarrow$ NextEvent($Q_{jobs}, t$)
10: **until** $t < H$
11: $\chi \leftarrow$ GroupTasks($\mathcal{X}, \Gamma$)
12: $\mathcal{S} \leftarrow$ GenerateScheduleTable($\chi$)
13: $\mathcal{V} \leftarrow$ GeneratePartitionTable($\chi$)
14: **return** $< \mathcal{O}, \mathcal{S}, \mathcal{V} >$

simulation is the set of schedules $\mathcal{S}$. In the simulation, we consider that the duration of each task is its WCET. With *EDF*, a task has the highest priority (and will be scheduled on its respective core) if its deadline $D_i$ comes earlier considering the current time. The outcome $S$ of a simulation is controlled by $\Theta$, the tasks offsets (their initial earliest activation) and $\Phi$, the relative deadlines used for each task in the simulation. These are modified for each task $\tau_i$ by our *SA* in Alg. 1, in the ranges $D_i$ to $T_i$ for the offsets $\Theta$ and 0 to $D_i$ for the relative deadlines $\Phi$. Our *EDF* simulation allows preemption (a higher priority task that is ready for execution may interrupt a lower priority task) considering the given macrotick, and can handle data dependencies, i.e. a task will not start before its predecessors have finished executing.

The *EDF* simulation is performed for the duration of a hyperperiod $H$ (which is also the system cycle), defined as the Least Common Multiple of all the task periods (line 1). SPH starts by creating a queue $Q_{jobs}$ with jobs of the tasks in $\Gamma$ that have to run during $H$ (line 2). Note that in our implementation these jobs are created on the fly, based on events occurring during the simulation. These events are generated by our simulation at design time, as part of the simulation used to derive the schedules. The simulation is performed in lines 4–10. Because SPH is run in each iteration of *SA*, we have optimized its implementation for speed, efficiency, simulation events and skipping only to events that have relevance for building the schedule tables $\mathcal{S}$. In the following, we explain how the simulation works in principle.

The simulation takes those jobs $\partial$ form $Q_{jobs}$ that are ready to execute at the time $t$ (line 5) and sorts them based on their priority (line 6). The job which has the earliest deadline and its precedent jobs are arrived, has the highest priority. The high-priority job is denoted with $\partial_H$. If the priority of $\partial_H$ is higher than the currently executing job, SPH preempts it and schedule $\partial_H$ instead (line 7). The simulation is stored in $\mathcal{X}$. SPH determines the next time in which a job becomes ready (line 9), considering the remainder of jobs in $Q_{jobs}$ and the macrotick parameter mentioned in Sect. II-A, which controls the granularity of preemption.

The final part of SPH post-processes the simulation data structure $\mathcal{X}$. SPH groups the time-wise consecutive jobs which have the same criticality level to form partitions, and also delays the tasks to insert the required task switching (in case preemptions were introduced) and partition overheads $\xi_i$ (line 11). See Sect. V-E for an illustration on how our heuristic works to create partitions by grouping tasks. SPH extracts the schedule tables $\mathcal{S}$ from the simulation $\chi$ (line 12), and the partition tables $\mathcal{V}$ (line 13).

### C. COST FUNCTION

In this section, we define the weighted cost function $\Omega$ in Eq.(6), used by our FCPC optimization strategy. The function has three terms (*QoC*, deviation of *QoC* and task schedulability constraint, respectively) and takes the *QoC* of control applications $J$ and the schedule tables $\mathcal{S}$ as input. The *QoC* optimization is controlled by the weights $\beta_1$ and $\beta_2$, whereas $\beta_3$ is a penalty value for the case when task deadlines are missed. The weights allow the system engineer to control the search for schedulable solutions that optimize *QoC*. Larger values for $\beta_1$ and $\beta_2$ will drive the search to optimize *QoC*, whereas a larger value for $\beta_3$ will drive the search faster to schedulable solutions.

$$\Omega = \beta_1 \times \bar{J} + \beta_2 \times \sigma_J + \beta_3 \times \Lambda \tag{6}$$

The control performance of control applications is captured by the first term. Assuming $m$ number of control applications, the average *QoC* for the applications is

$$\bar{J} = \frac{\sum_{i=1}^{m} J_i}{m}, \tag{7}$$

where, $J_i$ is the *QoC* for a control application $\gamma_i$ which is calculated by JitterTime and its value is mapped to the range [0, 1]. The range of cost performance is from 0, for the best-performance, to 1, for the worst-performance, e.g., which is unstable. $\bar{J}$ is normalized to the same range.

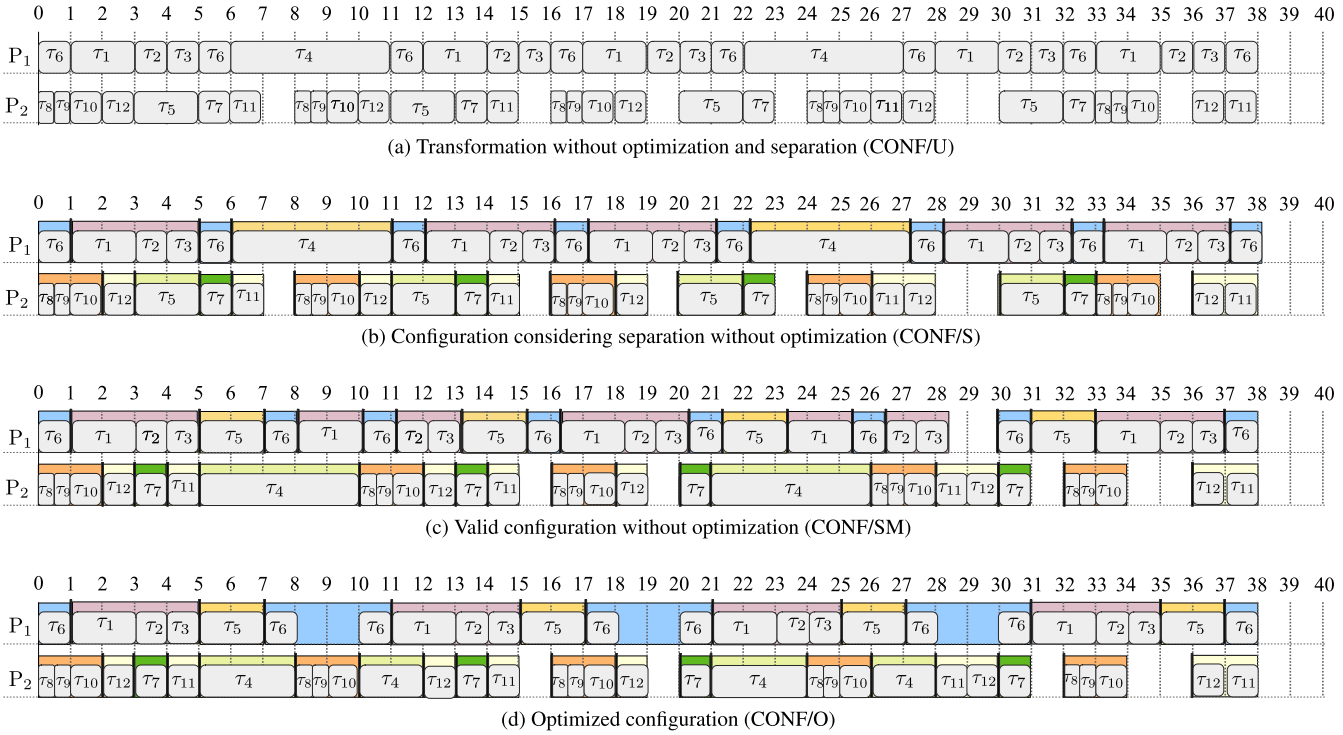
The second term captures the deviation among the *QoC* of the control applications, and is defined in Eq. (8). Concerning the range of $J$, the range of variation is from 0, for the equally distributed *QoC* of control applications, to $\sqrt{\frac{m-1}{m}} < 1$ for $m$ control applications when their performance costs are highly-deviated.

$$\sigma_J = \sqrt{\frac{\sum_{i=1}^{m} |J_i - \bar{J}|}{m}} \tag{8}$$

The last term is the function $\Lambda$, which is a constraint that checks for deadline violations for all the tasks in the schedule table. $\Lambda$ is also normalized, and starts form 0, for no deadline violations, to 1, for the case in which all the jobs have missed their deadlines.

### D. SA DESIGN TRANSFORMATIONS

As mentioned earlier, *SA* decide the mapping $\mathcal{M}$ of tasks to the core, assignment $\mathcal{O}$ of tasks to partitions, and periods $\mathcal{F}$ of control applications. The *SA* also varies the offsets of tasks




(a) Transformation without optimization and separation (CONF/U)

(b) Configuration considering separation without optimization (CONF/S)

(c) Valid configuration without optimization (CONF/SM)

(d) Optimized configuration (CONF/O)

**FIGURE 8.** **Four different configuration for the applications in Table. 2; The black lines show the overheads of the partitions: CONF/U has no temporal separation; CONF/S has several deadline misses; CONF/SM is feasible and the cost function value is 0.13522; CONF/O is the final solution which shows 77% improvement.**

and *EDF* deadline to create different scenarios for the *EDF* simulation, generating various schedules. *SA* uses *move*s to explore the solution space by generating randomly neighborhoods of the current solution. *SA* randomly selects one of the *move*s and applies it to randomly selected tasks to generate the neighborhood in each iteration. The *move*s are:

- **Swap Tasks**: swaps the mapping of two selected tasks.
- **Period Selection**: randomly chooses a period $T_i$ from the given set of periods $F_i$ for the selected control task $\tau_i$.
- **Deadline Adjustment**: randomly selects a relative deadline $\Phi$ (used in the *EDF* simulation in SPH) in the range from $D_i$ to $T_i$.
- **Offset Adjustment**: the offset $\Theta$ of the selected task $\tau_i$ is selected randomly in the interval from 0 to $D_i$.

To drive the search faster towards the schedulable solution with increased $QoC$, we encourage *SA* to pick tasks that need special attention, e.g., because they missed their deadline or they impact $QoC$. This is achieved by sorting the tasks based on the criteria we want to address (schedulability, $QoC$) and selecting randomly tasks based on probability density function that are skewed towards the head of the sorted list.

### E. ILLUSTRATIVE EXAMPLE FOR FCPC

Let us present an example illustrating how FCPC works. We have two cores, $P_1$ and $P_2$ and four applications, including two control applications $\gamma_1$ and $\gamma_4$. The applications have 12 tasks in total. Each control application is controlling an inverted pendulum in the upright position, with its process modeled as

$$G(s) = \frac{200}{s^2 + 400}. \tag{9}$$

Each of the control applications has three tasks which are respectively sensor, LQG controller, and actuator task, see Sect. IV-A. The controller tasks, $\tau_2$ in the control application $\gamma_1$, and $\tau_9$ in the control application $\gamma_4$ are LQG controllers which are designed using Jitterbug [28]. Table 2 shows the applications, tasks and their details.

FCPC starts with an initial configuration which comes from the initial solution (Sect. V-A, line 4 in Alg. 1). SPH uses this initial configuration and creates an *EDF* simulation (Alg. 2). We take the stored simulation $\mathcal{X}$ (line 7 in Alg. 2), group the task to create partitions (line 11 in Alg. 2) and generate a schedule table (line 12) and a partition table (line 13). Let us explain how partitions are create, starting from the schedule in Fig. 8a, which is the result of *EDF* simulation stored in $\mathcal{X}$. SPH post-processes the schedule from left to right, and, if two tasks share the same criticality levels, it group them into same partition. Let us call this configuration CONF/S (from configuration with separation) depicted in Fig. 8b, which shows a part of the schedule table starting from 0 ms to 40 ms. We use different colors to highlight the partitions in Fig. 8. For example, we create four partitions, denoted with $\delta_4$, $\delta_5$, $\delta_6$ and $\delta_7$ on core $P_2$, with criticality levels of $L_{\delta_4} = 1$, $L_{\delta_5} = 2$, $L_{\delta_6} = 3$ and $L_{\delta_7} = 0$.

**TABLE 2. Illustrative example applications.**

| Application | Tasks | L | WCET (ms) | T (ms) | D (ms) |
|---|---|---|---|---|---|
| $\gamma_1$ | $\tau_1$ | 3 | 2 | $F_1$ | 12 |
| | $\tau_2$ | 3 | 1 | $F_1$ | 12 |
| $F_1 = \{6, 8, 10, 12\}$ | $\tau_3$ | 3 | 1 | $F_1$ | 12 |
| $\gamma_2$ | $\tau_4$ | 1 | 5 | 20 | 20 |
| | $\tau_5$ | 1 | 2 | 10 | 10 |
| $\gamma_3$ | $\tau_6$ | 2 | 1 | 5 | 5 |
| | $\tau_7$ | 2 | 1 | 10 | 10 |
| $\gamma_4$ | $\tau_8$ | 3 | 0.5 | $F_4$ | 12 |
| | $\tau_9$ | 3 | 1 | $F_4$ | 12 |
| $F_4 = \{5, 8, 10, 12\}$ | $\tau_{10}$ | 3 | 0.5 | $F_4$ | 12 |
| $\gamma_5$ | $\tau_{11}$ | 0 | 1 | 12 | 12 |
| | $\tau_{12}$ | 0 | 1 | 9 | 9 |

The overhead times for these partitions are determined as 100 $\mu$s, 50 $\mu$s, 100 $\mu$s and 50 $\mu$s, respectively (we use the approach from [18], which considers the partition overheads of 5% of the largest task WCET in the partition). Regarding the task switching overheads, we use the values measured in [22]. SPH may delay the tasks to apply these overheads, hence several instances of tasks may miss their deadlines. For example, the task $\tau_3$ will miss its deadline at $t = 16$ ms for about 550 $\mu$s.

Let us consider that CONF/S is the current solution driving the search performed by *SA* in Alg. 1, and this *SA* perform a "Swap Task" design transformation in line. 8, which results in swapping the mapping of tasks $\tau_4$ and $\tau_5$. This will result in the configuration from Fig. 8c, which we call CONF/SM (from configuration with separation and mapping), which is feasible, i.e., there are no deadline misses. As a consequence of the task swapping, not only mapping of the tasks to the cores are swapped but also their assignments to the partitions, since they have the same criticality level. In this configuration, the control application $\gamma_1$ experiences maximum I/O jitter (12.5% of its period, which is 8 ms) and the control application $\gamma_4$ has no I/O jitter (release and start jitters are seen at $t = 10$ ms and $t = 16$ ms). The cost of control is calculated by JitterTime, and the average of the two control applications is 0.09642 and the deviation is equal to 0.0388. The cost function, calculated as in Eq. (6) considering a value of 1 for all weights, has a value of 0.13522.

*SA* will accept the configuration in Fig. 8c as the current solution, since it improves over Fig. 8b that had deadline misses (line 13 in Alg. 1). Let us assume that the next design transformation is done by "Period Selection", e.g., by selecting the period of 10 ms for tasks in the control application $\gamma_1$. The resulted optimized configuration (CONF/O) is depicted in Fig. 8d. The values of average *QoC* and the deviations (terms one and two in Eq. (6)) are 0.0268 and 0.0033, respectively, resulting in a cost function of 0.0301, which is an improvement of 77% over the CONF/SM in Fig. 8c.

## VI. EXPERIMENTAL EVALUATION

Our proposed optimization strategy, Fog Computing Platform Configuration (FCPC), was implemented in C#, and all the experiments were run on a laptop with an i7 CPU at 3.0 GHz and 32 GB of RAM. We investigate the performance of our proposed method on ten test cases, which have mixed-criticality tasks. The details of test cases are shown in Table 3, where column 2 shows the total number of cores in the *FCP*, column 3 shows the total number of control applications, column 4 shows the total number of tasks, and column 5shows the total number of tasks having a particular criticality level, 0 to 4. Each test case has multiple control applications: Each control application has three control tasks (see Sect, IV-A) and the control task which implements the controller is a LQG controller designed with Jitterbug to control a plant using one of the three different processes which are defined in Eq. (9), Eq. (10), and Eq. (11),

$$G(s) = \frac{300}{s^2 - 200}, \tag{10}$$

$$G(s) = \frac{100}{s^2 + 300}. \tag{11}$$

The tasks in each test case represent real-time tasks with different criticality levels and can be run on any of the cores. Tasks with the same criticality level are mapped to the same partition and overheads are applied to each partition slice.

The results of evaluation are presented in Table 3. The $\Omega$ columns show the cost function of test cases for each solution. The results obtained by running FCPC on each test cases are reported in column 6 using the value of the cost function $\Omega$. We have set the weights $\beta_1$, $\beta_2$ and $\beta_3$ to 0.45, 0.1 and 1.0, respectively. The weights were determined experimentally to guide the search faster towards solutions with optimized *QoC*. $\beta_1$ can be set by analyzing the stability of the control applications with Jitterbug and choosing a value that drives the search towards stable control. Jitterbug also reports the phase margins (smaller phase margin means larger sensitivity) of the applications and $\beta_2$ is set to allow a larger deviation from the mean *QoC* if there is a large variation among the phase margins of the applications. A $\beta_3$ value of 1.0 is a relatively large penalty value considering that the cost function terms are in the range [0, 1]. To determine the ability of FCPC to improve the *QoC* measured by $\Omega$, Table 3 also reports the results obtained by three variants of FCPC, as follows:

- *FCPC/M*: does not optimize the mapping of tasks and uses the mapping determined in the initial solution, as explained in Sect. V-D.
- *FCPC/Q*: does all the optimizations of FCPC but does not use the *QoC* in the cost function (the first two terms), hence it optimizes only for schedulablity, ignoring the control performance.
- *FCPC/P*: generates solutions with our proposed strategy without considering period selection for critical control tasks. The period of control tasks are set to their smallest value.

The other $\Omega$ columns show the value in terms of percentage in deterioration of the cost function for FCPC/M, FCPC/Q and FCPC/P, respectively, compared to FCPC. A larger cost

**TABLE 3.** Evaluation results for our proposed optimization.

| Test cases | No. of Cores | Total No. of Control Applications | Total No. of Tasks | Total No. of Tasks for Criticality level of [0-4] | Ω of FCPC | Ω for FCPC/M | Ω for FCPC/Q | Ω for FCPC/P |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 12 | {1,2,2,1,6} | 0,19 | 48% | Not Feasible | 60% |
| 2 | 2 | 3 | 23 | {3,2,5,4,9} | 0,34 | 5% | Not Feasible | Not Feasible |
| 3 | 2 | 2 | 17 | {2,2,4,3,6} | 0,21 | 5% | Not Feasible | 33% |
| 4 | 2 | 2 | 23 | {2,1,7,6,7} | 0,29 | 13% | Not Feasible | 13% |
| 5 | 3 | 3 | 32 | {1,4,8,8,11} | 0,21 | 39% | Not Feasible | 55% |
| 6 | 3 | 3 | 31 | {2,3,9,7,10} | 0,22 | 85% | Not Feasible | 50% |
| 7 | 3 | 4 | 33 | {4,4,8,7,12} | 0,26 | 15% | Not Feasible | 10% |
| 8 | 4 | 4 | 44 | {4,6,11,9,14} | 0,20 | 29% | Not Feasible | 72% |
| 9 | 5 | 6 | 54 | {7,5,14,10,18} | 0,21 | 21% | Not Feasible | Not Feasible |
| 10 | 6 | 7 | 63 | {6,7,16,12,22} | 0,24 | 21% | Not Feasible | 46% |

function value, i.e., larger percentage deterioration, means a worse-quality solution.

As we can see from Table 3, FCPC has been able to obtain feasible solutions for all the test cases, i.e., all the tasks' deadlines are satisfied and all the controllers are stable and have good *QoC*. The average value of cost function Ω is 0.24 and the values are not highly-deviated in all the test cases. We have used a time limit of 20 to 70 minutes as a termination criteria for FCPC and its variants, depending on the size of the test case.

When comparing FCPC with its variants that ignore certain optimization aspects, we can see in the last three columns of Table 3 large percentage deterioration, or even "unstable" control applications (shown with "Not Feasible") for nearly all the test cases. For example, the results show that not considering *QoC* in FCPC/Q gives the worst results, which demonstrates that the JITTERTIME-based *QoC* evaluation of solutions needs to be used during the optimization of a *FCP* configuration. Otherwise, the controllers become unstable even though the deadlines are not missed. Determining the right period for the control applications is very important as we can see in case of FCPC/P, where considering a single period results in control applications that are unstable and the degradation is high, on average 42% in all other cases. The results also show the importance of mapping, since using in FCPC/M, the mapping determined by the initial solution, the degradation is on the average 28%.

### A. REALISTIC TEST CASE

We have evaluated the proposed optimization strategy on a realistic test case which consists of 8 applications running on a fog node inside a vehicle. Future vehicles are envisioned to be "fog nodes on wheels" [33] as they integrate more and more functions and become interconnected with each other.

The details of the test case are in Table 4. We have 8 applications running on a dual-core fog node which include a drive-assistance application for radar-cruise control (application $\gamma_4$). The car is modeled with a first-order transfer function and the controller is a LQG speed controller which is design by Jitterbug. Application $\gamma_1$ monitors the engine, $\gamma_2$ is a passenger comfort application that controls the climate,

**TABLE 4.** Realistic test case.

| Application | Tasks | L | WCET (ms) | T (ms) | D (ms) |
|---|---|---|---|---|---|
| $\gamma_1$ | $\tau_1$ | 2 | 0.5 | 10 | 10 |
| | $\tau_2$ | 2 | 0.5 | 10 | 10 |
| | $\tau_3$ | 2 | 1 | 10 | 10 |
| $\gamma_2$ | $\tau_4$ | 1 | 2 | 15 | 15 |
| | $\tau_5$ | 1 | 2 | 15 | 15 |
| | $\tau_6$ | 1 | 1.5 | 15 | 15 |
| | $\tau_7$ | 1 | 1 | 15 | 15 |
| | $\tau_8$ | 1 | 2 | 15 | 15 |
| $\gamma_3$ | $\tau_9$ | 2 | 1 | 20 | 20 |
| | $\tau_{10}$ | 2 | 0.5 | 20 | 20 |
| | $\tau_{11}$ | 2 | 0.5 | 20 | 20 |
| | $\tau_{12}$ | 2 | 1 | 20 | 20 |
| | $\tau_{13}$ | 2 | 1.5 | 20 | 20 |
| | $\tau_{14}$ | 2 | 0.5 | 20 | 20 |
| $\gamma_4$ | $\tau_{15}$ | 3 | 0.5 | $F_4$ | 24 |
| | $\tau_{16}$ | 3 | 2.5 | $F_4$ | 24 |
| | $\tau_{17}$ | 3 | 3 | $F_4$ | 24 |
| | $\tau_{18}$ | 3 | 1 | $F_4$ | 24 |
| | $\tau_{19}$ | 3 | 3 | $F_4$ | 24 |
| | $\tau_{20}$ | 3 | 1 | $F_4$ | 24 |
| $F_4 = \{18, 20, 22, 24\}$ | $\tau_{21}$ | 3 | 0.5 | $F_4$ | 24 |

$\gamma_3$ is used for image analysis as part of driver-assistance functionality. We give the applications different critically levels, based on their importance, as presented in the table.

Our proposed optimization strategy has successfully scheduled all the tasks and decided the task mapping to the partitions and cores. The results show that none of the tasks has missed its deadline. Furthermore, the mapping of tasks to the cores shows the core utilization of 86.67% and 86.88% for the dual-core processor. We used JITTERTIME to simulate the controller behavior and calculate the cost of control for the application concerning the given cost function in IV-C with the weights of 0.25, and 0.25 and 1 for $\beta_1$, $\beta_2$ and $\beta_3$. The cost function has the value of 0.007.

## VII. RELATED WORK

There is already much work on various topics related to Fog Computing [34]–[36]. Even though basic quality-of-service (*QoS*) for applications has been addressed, the *QoC* for control applications in the fog is still an open issue. However, there is a lot of useful literature in works that tackle the problem of degradation of control applications [37]–[39].

Researchers propose several approaches (such as partial and spatial separation of control tasks, virtualization of PLCs, scheduling of control tasks, co-design of control applications) that guarantee extra-functional properties of control applications [40]–[43]. The presented approaches are well studied and categorized into a category for platform configuration and a category for the integration of applications. Good performance for control applications will be ensured if both the applications and the platforms are configured.

Separation and isolation of applications regarding criticality levels ensure resource allocation to control applications. Researchers propose spatial and temporal separation to integrate mixed-criticality applications [39], [42], [43]. In this approach, partitions with different criticality levels separate the application. Resources are allocated to the partitions based on the criticality level, which assures resource availability and accessibility for critical applications with high priority. These applications would have guaranteed dependability by promising separation. A presentation of scheduling in mixed-criticality systems, which also considers partitioning, is presented in [44]. For example, Tamas-Selicean *et al.* [45] propose a method in which different Safety-Integrity Levels (SIL) are assigned to the applications. In this method, applications with the same SIL are mapped to a single partition. Each partition is allocated several time slots on a processor to execute respective tasks. Concerning this method, the approach can provide a partition for each control application.

On the other hand, integration of the applications in the platform affects their functional and extra-functional properties. The co-design of control applications configure them at integration level to achieve the highest performance. The co-design approach takes the platform characteristics into account while designing the application to have good integration with the platform.

The *QoC* analysis and schedulability of the tasks are taken into account while designing the control applications in [46] and [47]. In the proposed approaches, the task scheduler schedules the tasks concerning the *QoC* of control applications. Besides, co-design is used to determine the period of tasks and design a robust and optimal controller. Other researchers also focused on the co-design and scheduling of control tasks to achieve the maximum *QoC* for control applications [39], [42], [43]. Co-design and scheduling concerning *QoC* for control applications are also proposed in the seminal work of Seto et al [26]. The authors optimize the period of control applications concerning the *QoC* and schedulability of the tasks.

Chwa *et al.* [48] propose a co-design and scheduling method to maximize *QoC* for control applications. The authors assign a sampling period, and a maximum number of consecutive deadline misses as parameters for each task concerning system stability. Then, the tasks are scheduled concerning the parameters without compromising system stability and also with efficient use of resources. Mahmoud *et al.* [43] use optimization algorithms to derive a timing constraint of control tasks such as the task period to achieve maximum *QoC* for the control applications concerning the schedulability of the control tasks [49]. In this work, heuristic algorithms are used to derive the period of tasks, deadlines of the tasks and end-to-end response of the control loop. The assigned parameters are assessed in a simulation that schedules and executes the tasks.

Samii *et al.* [50] present an approach in which a controller is synthesized for each plant, and the control tasks are scheduled concerning the priority of the tasks. In this work, the scheduling is based on the cost of control function, which aims to consider the maximum *QoC* for all the control applications. The same approach concerning co-design and scheduling of control application is used in [51]. A similar co-design approach is presented by Cervin *et al.* [52]. In this work, the scheduler uses feedback from execution time and also feed-forwards the workload along with the cost of control to achieve the best *QoC*. Besides, control task parameters such as period are changed with the feedback from execution time. The approach can compensate the impact of jitter on the *QoC*.

Task scheduling has a significant impact on the performance of control applications. The *QoC*-aware scheduling reduces the degradation of control applications to some levels based on the criticality of the application. Configuration of the platform at the computation level, especially in the task scheduler guarantees good performance for control applications. Barzegaran *et al.* [21] have presented a heuristic approach for scheduling of tasks and mapping them to the cores which maximizes the *QoC* of control applications. The work also shows that allowing preemption in scheduling of tasks improves the schedulability of tasks and *QoC* of control applications. The work does not consider separation of mixed-criticality tasks which is covered in this paper, and it also ignores the effect of control tasks periods on the schedulability, which is also covered in this paper. Task period selection and cost function definition for the optimal control behavior is based on the cost of the control. The scheduling aims to cover the bounded jitter and latency regarding the stability margin of the control applications. In the work by Schneider *et al.* [53], the *QoC* measurement is embedded in the task scheduler with allowed preemption. The scheduler is capable of handling mixed-criticality applications as well.

Another co-design approach is considered in [54]. The authors get feedback of delay and jitter in the execution of tasks from the scheduler and feed it to the control applications. The controller takes the feedback and adjusts the control output to compensate the delay and jitter impact and to maximize the *QoC*. The feedback from the task scheduler is also used to predict the jitter and delay. In [55], a feedback scheduling framework is developed to schedule control tasks such that the *QoC* is maximized for control applications and to adjust workload constraints. The *QoC* measurement is embedded in the task scheduler. The scheduler gets delay and jitter feedback to change the period of the tasks concerning the *QoC* and workload management. The same approach is used in [56], to schedule tasks concerning the *QoC* with

feedback from scheduling. In this work, the period of tasks is changed regarding the feedback. Eker *et al.* [57], propose a similar method. The method uses feedback from the scheduler to assign the period of control tasks. The period assignment provides good control performance along with optimizing the resource allocation of the task.

Cha *et al.* [58] propose a method for scheduling of control tasks which determines the deadline and period of the tasks for achieving maximum *QoC*. The method optimizes the *QoC* of the control applications and resource utilization. In co-design approach presented in [59], the task scheduler guarantees bounded delay and jitter in execution of control application while the co-design approach guarantees that the control application is still stable in the presence of bounded delay and jitter. In work by Fan *et al.* [60], a scheduling algorithm is proposed that can provide some degree of isolation, which can host control applications. In this work, control applications can be assigned to partitions to ensure the separation. The algorithm also maps the tasks to the cores.

## VIII. CONCLUSION AND FUTURE WORK

In this paper, we have addressed the problem of configuring the mapping, partitioning, scheduling and periods of mixed-criticality tasks when implementing the applications on a Fog Computing Platform. The optimized solution has good and balanced Quality-of-Control for critical control applications, ensuring the schedulability of all real-time tasks, as well as spatial and temporal isolation for mixed-criticality tasks. Our proposed strategy is based-on a Simulated Annealing meta-heuristics, which uses an Earliest Deadline First simulation.

We have evaluated this strategy on several test cases. As the results show, our proposed optimization strategy successfully generates solutions which have good and balanced Quality-of-Control for control applications considering temporal isolation for all the test cases in comparison with the solutions that have ignored some of the optimization criteria.

The successful virtualization of control, achieving the same control performance (and dependability) as the one taken for granted in OT, is a crucial step towards the adoption of Fog Computing in the industrial area. In our future work, we will consider the effect of the communication; we will take into account the possibility of incremental scheduling based on our proposed strategy, and we will also consider other optimization techniques such as constraint programming to solve the problem.

## REFERENCES


[1] M. García-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *J. Syst. Archit.*, vol. 60, no. 9, pp. 726–740, Oct. 2014.

[2] O. Consortium. (2017). *OpenFog Reference Architecture for Fog Computing*. Accessed: Jan. 5, 2020. [Online]. Available: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17%.pdf

[3] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for Internet of Things and analytics," in *Big Data Internet Things: A Roadmap for Smart Environments*. Cham, Switzerland: Springer, 2014, pp. 169–186.

[4] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog computing for the Internet of Things: A survey," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–41, 2019.

[5] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: Architecture, key technologies, applications and open issues," *J. Netw. Comput. Appl.*, vol. 98, pp. 27–42, Nov. 2017.

[6] TTTech Computertechnik AG. (2019). *Nerve*. Accessed: Oct. 7, 2019. [Online]. Available: http://tttech.com/products/industrial/industrial-iot/nerve

[7] ACRN. (2019). *Official Website of the Project ACRN*. Accessed: Dec. 8, 2019. [Online]. Available: http://projectacrn.org/

[8] R. Kaiser and S. Wagner, "The pikeos concept: History and design," SysGO AG, Mainz, Germany, White Paper, 2007. [Online]. Available: http://www.sysgo.com

[9] IEEE. (2016). *Official Website of the 802.1 Time-Sensitive Networking Task Group*. Accessed: Dec. 8, 2019. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html

[10] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst. RTNS*, 2016, pp. 183–192.

[11] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. Workshop Mobile Big Data*, Jun. 2015, pp. 37–42.

[12] FORA. (2019). *Fog Computing Platform: Requirements and Initial Designs*. Accessed: Oct. 7, 2019. [Online]. Available: http://www.fora-etn.eu/deliverables/

[13] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," SRI Int. MENLO Park CA Comput. Sci. Lab., Washington, DC, USA, Tech. Rep. DOT/FAA/AR-99/58, 2000.

[14] The Linux Foundation. (2019). *Xen Project*. Accessed: Oct. 7, 2019. [Online]. Available: https://xenproject.org

[15] Universidad Politécnica de Valencia. (2019). *XtratuM Hypervisor*. Accessed: Jan. 19, 2020. [Online]. Available: https://xtratum.org

[16] N. R. Storey, *Safety Critical Computer Systems*. Reading, MA, USA: Addison-Wesley, 1996.

[17] S. Xi, M. Xu, C. Lu, L. T. X. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in xen," in *Proc. 14th Int. Conf. Embedded Softw. - EMSOFT*, 2014, pp. 1–10.

[18] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Scheduling optimization with partitioning for mixed-criticality systems," *J. Syst. Archit.*, vol. 98, pp. 191–200, Sep. 2019.

[19] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms Applications*, vol. 24. Boston, MA, USA: Springer, 2011.

[20] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS )*, Dec. 2007, pp. 239–243.

[21] M. Barzegaran, A. Cervin, and P. Pop, "Towards quality-of-control-aware scheduling of industrial applications on fog computing platforms," in *Proc. Workshop Fog Comput. IoT*, Apr. 2019, pp. 1–5.

[22] S. S. Craciunas, R. S. Oliver, and V. Ecker, "Optimal static scheduling of real-time tasks on distributed time-triggered networked systems," in *Proc. IEEE Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2014, pp. 1–8.

[23] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of scheduling techniques for addressing shared resources in multicore processors," *ACM Comput. Surveys*, vol. 45, no. 1, pp. 1–28, Nov. 2012.

[24] K. Ogata and Y. Yang, *Modern Control Eng.*, vol. 4. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[25] K. Astrom and B. Wittenmark, *Computer Controlled System*. Upper Saddle River, NJ, USA: Prentice-Hall, 1997.

[26] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proc. 17th IEEE Real-Time Syst. Symp.*, Dec. 1996, pp. 13–21.

[27] M. Krstic, *Systems & Control: Foundations and Applications Delay Compensation for Nonlinear, Adaptive, and PDE Systems*. Basel, Switzerland: Birkhäuser Boston, 2009.

[28] B. Lincoln and A. Cervin, "JITTERBUG: A tool for analysis of real-time control performance," in *Proc. 41st IEEE Conf. Decis. Control*, Dec. 2002, pp. 1319–1324.

[29] A. Cervin, P. Pazzaglia, M. Barzegaran, and R. Mahfouzi, "Using JitterTime to analyze transient performance in adaptive and reconfigurable control systems," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2019, pp. 1025–1032.

[30] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.

[31] E. K. Burke and G. Kendall, *Search Methodologies*. Boston, MA, USA: Springer, 2005.
[32] O. Sinnen, *Task Scheduling for Parallel Systems*, vol. 60. Hoboken, NJ, USA: Wiley, 2007.
[33] M. Chiang, B. Balasubramanian, and F. Bonomi, *Fog for 5G IoT*, vol. 288. Hoboken, NJ, USA: Wiley, 2017.
[34] W. Yu, F. Liang, X. He, W. Grant Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.
[35] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*. Singapore: Springer, 2018, pp. 103–130.
[36] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-Art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 416–464, 1st Quart., 2018.
[37] X.-M. Zhang, Q.-L. Han, and X. Yu, "Survey on recent advances in networked control systems," *IEEE Trans. Ind. Informat.*, vol. 12, no. 5, pp. 1740–1752, Oct. 2016.
[38] D. Zhang, P. Shi, Q.-G. Wang, and L. Yu, "Analysis and synthesis of networked control systems: A survey of recent advances and challenges," *ISA Trans.*, vol. 66, pp. 376–392, Jan. 2017.
[39] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proc. IEEE*, vol. 95, no. 1, pp. 138–162, Jan. 2007.
[40] D. Simon, A. Seuret, and O. Sename, "Real-time control systems: Feedback, scheduling and robustness," *Int. J. Syst. Sci.*, vol. 48, no. 11, pp. 2368–2378, Aug. 2017.
[41] W. Zhang, M. S. Branicky, and S. M. Phillips, "Stability of networked control systems," *IEEE Control Syst. Mag.*, vol. 21, no. 1, pp. 84–99, Feb. 2001.
[42] Z.-W. Wang and H.-T. Sun, "Control and scheduling co-design of networked control system: Overview and directions," in *Proc. Int. Conf. Mach. Learn. Cybern.*, Jul. 2012, pp. 816–824.
[43] M. S. Mahmoud and M. M. Hamdan, "Fundamental issues in networked control systems," *IEEE/CAA J. Automatica Sinica*, vol. 5, no. 5, pp. 902–922, Sep. 2018.
[44] A. Burns and R. Davis, "Mixed criticality systems—A review," Dept. Comput. Sci., Univ. York, York, U.K., Tech. Rep., 2013, pp. 1–69.
[45] D. Tămaş-Selicean and P. Pop, "Design optimization of mixed-criticality real-time embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 3, pp. 50–55, Apr. 2015.
[46] K.-E. Arzen, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *Proc. 39th IEEE Conf. Decis. Control*, Dec. 2000, pp. 4865–4870.
[47] Y. Xu, A. Cervin, and K.-E. Arzen, "Harmonic scheduling and control co-design," in *Proc. IEEE 22nd Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2016, pp. 182–187.
[48] H. S. Chwa, K. G. Shin, and J. Lee, "Closing the gap between stability and schedulability: A new task model for cyber-physical systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2018, pp. 327–337.
[49] M. Ryu, S. Hong, and M. Saksena, "Streamlining real-time controller design: From performance specifications to end-to-end timing constraints," in *Proc. 3rd IEEE Real-Time Technol. Appl. Symp.*, Jun. 1997, pp. 91–99.
[50] S. Samii, A. Cervin, P. Eles, and Z. Peng, "Integrated scheduling and synthesis of control applications on distributed embedded systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 57–62.
[51] R. Majumdar, I. Saha, and M. Zamani, "Performance-aware scheduler synthesis for control systems," in *Proc. 9th ACM Int. Conf. Embedded Softw. EMSOFT*, 2011, pp. 299–308.
[52] A. Cervin, J. Eker, B. Bernhardsson, and K. E. Årzén, "Feedback-feedforward scheduling of control tasks," *Real-Time Syst.*, vol. 23, nos. 1–2, pp. 25–53, 2002.
[53] R. Schneider, D. Goswami, A. Masrur, and S. Chakraborty, "QoC-oriented efficient schedule synthesis for mixed-criticality cyber-physical systems," in *Proc. Forum Specification Design Lang.*, Sep. 2012, pp. 60–67.
[54] Z. Sahraoui, E. Grolleau, D. Mehdi, M. Ahmed-Nacer, and A. Labed, "Predictive-delay control based on real-time feedback scheduling," *Simul. Model. Pract. Theory*, vol. 66, pp. 16–35, Aug. 2016.
[55] Y.-C. Tian and L. Gui, "QoC elastic scheduling for real-time control systems," *Real-Time Syst.*, vol. 47, no. 6, pp. 534–561, Dec. 2011.
[56] Z. Sahraoui, E. Grolleau, M. A. Nacer, D. Mehdi, and H. Bauer, "Antinomy between schedulability and quality of control using a feedback scheduler," in *Proc. 22nd Int. Conf. Real-Time Netw. Syst. - RTNS*, 2014, pp. 171–179.
[57] J. Eker, P. Hagander, and K.-E. Årzén, "A feedback scheduler for real-time controller tasks," *Control Eng. Pract.*, vol. 8, no. 12, pp. 1369–1378, Dec. 2000.
[58] H.-J. Cha, W.-H. Jeong, and J.-C. Kim, "Control-scheduling codesign exploiting trade-off between task periods and deadlines," *Mobile Inf. Syst.*, vol. 2016, pp. 1–11, Apr. 2016.
[59] Y. Xu, A. Cervin, and K.-E. Arzen, "Jitter-robust LQG control and real-time scheduling co-design," in *Proc. Annu. Amer. Control Conf. (ACC)*, Jun. 2018, pp. 3189–3196.
[60] M. Fan and G. Quan, "Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 503–508.

**MOHAMMADREZA BARZEGARAN** has been a Marie Curie Ph.D. Fellow of computer science with the Technical University of Denmark, since 2018. His research is focused on the configuration of fog computing platform for critical control applications. His main research interests include optimization, the configuration of real-time and safety-critical systems, and co-design of control applications for real-time and safety-critical systems.

**ANTON CERVIN** (Member, IEEE) received the M.Sc. degree in computer science and technology and the Ph.D. degree in automatic control from Lund University, Sweden, in 1998 and 2003, respectively. He is currently an Associate Professor with the Department of Automatic Control, Lund University, where he does research on event-based estimation and control, autonomous real-time systems, and controller-scheduling co-design.

**PAUL POP** (Member, IEEE) received the Ph.D. degree in computer systems from Linköping University, in 2003. He has been an Associate Professor with the DTU Compute, Technical University of Denmark, and since 2016, he has also been a Professor of cyber-physical systems. His research is focused on developing methods and tools for the analysis and optimization of dependable embedded systems. In this area, he has published over 130 peer-reviewed articles, 3 books, and 7 book chapters. His research has been highlighted as The Most Influential Papers of 10 Years DATE. He has served as a Technical Program Committee member on several conferences, such as DATE and ESWEEK. He received the best paper award at DATE 2005, RTIS 2007, CASES 2009, MECO 2013, and DSD 2016. He also received the EDAA Outstanding Dissertations Award (Co-Supervisor), in 2011. He is also the Chairman of the IEEE Danish Chapter on Embedded Systems. He is also the Director of the DTU's IoT Research Center and has coordinated the Danish National InfinIT Safety-Critical Systems Interest Group.