Discrete Event Dynamic Systems
https://doi.org/10.1007/s10626-024-00407-0

Tracking dynamic deadlines in switched max-plus linear systems with uncontrollable workloads

Roohallah Azarmi[1] · Mohsen Alirezaei[2,3] · Dip Goswami[1] · Twan Basten[1]

Received: 17 October 2022 / Accepted: 22 October 2024

Abstract
Modern safety-critical cyber-physical systems such as medical imaging equipment or autonomous vehicles need to respect strict deadlines on received data-processing workloads. These deadlines and workloads are dynamic and uncontrollable and the systems typically have only a limited discrete number of system configurations to respond to dynamic changes. The number and types of processors allocated to a data-processing task, their operating voltage and frequency, and the resolution and frequency of sensing (e.g., images) are examples of controllable configuration parameters. Guaranteeing dynamically changing deadlines under uncontrollable workloads with a limited discrete number of response options can be phrased as a multi-objective tracking problem for a switched max-plus linear system. This results in a combined scheduling and control problem. We propose an integrated state-feedback and model-predictive control solution that minimizes the number of deadline misses and the cost of implementation (e.g., energy consumption). We demonstrate the effectiveness of our approach through simulation.

Keywords Safety-critical cyber-physical systems · Dynamic deadlines · Uncontrollable workloads · Switched max-plus linear systems · MPC

✉ Roohallah Azarmi
r.azarmi@tue.nl; R.Azarmi.2016@ieee.org

Mohsen Alirezaei
m.alirezaei@tue.nl

Dip Goswami
d.goswami@tue.nl

Twan Basten
a.a.basten@tue.nl

[1] Electronic Systems Group, Department of Electrical Engineering, Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands

[2] Department of Mechanical Engineering, Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands

[3] RTD Department, Siemens Industry Software and Services B.V., Helmond, The Netherlands

1 Introduction

Data is a key resource in today's modern world (Bajaber et al. 2016). *Data-processing systems* and applications play a pivotal role in smart society, e.g., in buildings, transportation, health-care systems, or security. Data-intensive sensing and control is rapidly gaining importance (Van Horssen 2018). Prime examples of successful applications are surgical robotics (Oda et al. 2009) and visual navigation (Chakraborty et al. 2016). Optimizing and controlling the performance of data-processing systems is still a challenging task (Chen and Zhang 2014; Fahad et al. 2014; Zhang et al. 2018; Stonebraker et al. 2005).

Safety-critical cyber-physical systems need to respect strict deadlines on processing data workloads. The processing deadlines may dynamically change over time and the data processing times depend on the content of the data as well as on the resources available for processing (Gheorghita et al. 2009). For instance, in autonomous driving, processing deadlines may vary with the speed of the vehicle and timing requirements of feedback controllers. The processing workload may depend on for instance the activated functionality (adaptive cruise control, lane-keeping assist) and the number of relevant features (cars, lane markers) in the sensor data. The time needed to process a given workload on an embedded compute platform depends on the number, type, and speed of available processing cores. Dimensioning a system for the worst case (highest workload and minimum deadline) is generally expensive (Van Horssen 2018; Liroz-Gistau et al. 2013), e.g., in terms of the number of required embedded processing units and energy consumption. To *meet dynamic deadlines under uncontrollable dynamic workloads while optimizing implementation cost* is a challenging problem. *The system configuration*, such as resolution and rate of sensors and type, number, and speed of the allocated processing units, *needs to be continuously adapted to fit the situation at hand*. An important aspect is that the system often has only *a limited, discrete number of configuration options* to respond to dynamic changes.

Meeting dynamic deadlines in a system with uncontrollable dynamic workloads while minimizing implementation cost can be seen as a multi-objective *tracking problem*. *Feedback control* is a systematic way to deal with tracking problems (Doyle et al. 2013). A variety of solutions for traditional linear systems exists, e.g., proportional-integral-derivative (PID) control (Aström and Hägglund 2006), state-feedback control (Dorf and Bishop 2008), and model-predictive control (MPC) (Camacho and Alba 2013). However, tracking with only a limited number of discrete options to respond to dynamic changes is challenging.

A means to cope with the combination of uncontrollable dynamic workloads and limited system configuration options is the use of *system scenarios* (Gheorghita et al. 2009). A number of well-defined modes of operation, the system scenarios, can be identified at design time, determined for instance by data content and/or system configuration, with nominal processing times per scenario. The system can then be appropriately configured at run-time. In this way, our tracking problem reduces to *an integrated scheduling and control problem*. *Selecting a system configuration* for each received workload is the control part; *deciding on the time to start the processing* of the workload is the scheduling part. The solution needs to take into account the characteristics of the workload, its processing deadline, time and cost of processing for possible system configurations, and time and cost for switching between configurations.

Max-plus algebra (MPA) is well suited to model and analyze timing aspects of discrete-event systems with a basis similar to the domain of linear systems (Baccelli et al. 1992; Komenda et al. 2018; Hardouin et al. 2018). A *switched max-plus linear system (SMPLS)* consists of subsystems that each are a max-plus linear system. SMPLS are a suitable means to model, analyze, schedule and control timing aspects of systems with different modes of operation (Van Den Boom and De Schutter 2012). Each system configuration is captured by a subsystem in the SMPLS. Many real-life systems can be modeled through SMPLS, such as production systems (Van Den Boom and De Schutter 2006; Basten et al. 2020), traffic and railway networks (Van Den Boom and De Schutter 2006), and data-processing systems (Geilen et al. 2020). SMPLS also form a suitable basis for the scheduling and control tracking problem considered in this paper.

An SMPLS model typically assumes constant timing delays. Data-processing times on modern processors, however, may suffer from small variations (relative to the coarse-grained variations captured in the different workloads), depending for instance on the state of data caches or small differences in the data content within a given workload. Such uncertainties in data-processing times with respect to nominal processing times may be captured in an SMPLS in max-plus-multiplicative form (Van Den Boom and De Schutter 2002).

This brings us to the following **problem definition:**

How to minimize deadline misses and implementation cost for a system with a limited number of discrete system configurations under uncontrollable dynamic workloads and dynamic deadlines in the presence of bounded model uncertainties?

The problem definition draws inspiration from data-processing systems, as elaborated above. However, the solutions presented in this paper apply to any system that can be captured as an SMPLS and satisfies the problem definition.

The **contributions** of this paper are as follows: We propose a control method for tracking and optimizing performance of a class of systems with uncontrollable varying workloads and dynamic workload deadlines, subject to input constraints in the form of a discrete number of configuration options that are available to respond to dynamic changes.

1. The targeted class of systems is cast into the framework of SMPLS and respective structural properties are provided.
2. The proposed control method integrates MPC to select optimal sybsystems at run-time in response to received workloads and state-feedback control to schedule the switching and to cope with the model uncertainties. It takes into account the time and cost of switching.
3. The method is shown to satisfy tracking requirements, minimizing deadline misses and optimizing the cost of implementation (e.g., energy consumption), enhancing robustness against model uncertainties, uncontrollable dynamic workload variations, and dynamically changing deadlines.

We review related work in Section 2. Section 3 introduces a motivating case study that serves as running example. Section 4 gives max-plus preliminaries. The problem setting is made precise in Section 5. Section 6 presents the control method. Section 7 shows its effectiveness through simulation. Section 8 concludes.

2 Related work

Several classes of performance control problems in the max-plus domain are covered in literature: optimization, tracking, and regulation problems. Optimization problems concern maximizing the productivity of output per unit of time, e.g., van den Boom et al. (2020); Alirezaei et al. (2012); Fusco et al. (2018); Brunsch et al. (2012); Nasri et al. (2014). Tracking problems, as the problems considered in this paper and in Dirza et al. (2019); Menguy et al. (2000); Schafaschek et al. (2020); Lahaye et al. (2008); Silva and Maia (2016), focus on dynamic set-point tracking. Finally, regulation problems, e.g., Aberkane et al. (2021); Goncalves et al. (2017, 2015), cover a specific class of tracking problems where the timing constraint (set-point) to be tracked is fixed.

The control solutions proposed for problems in the max-plus domain can be categorized into four groups: MPC (van den Boom et al. 2020; Alirezaei et al. 2012; Dirza et al. 2019), state-feedback control (Fusco et al. 2018; Brunsch et al. 2012), constraint-based optimal control (Menguy et al. 2000; Schafaschek et al. 2020; Lahaye et al. 2008; Silva and Maia 2016), and nonlinear optimization (Nasri et al. 2014). The techniques developed in Dirza et al. (2019); Menguy et al. (2000); Lahaye et al. (2008); Silva and Maia (2016); Goncalves et al. (2017, 2015) for max-plus regulation and tracking problems all have continuous configuration control variables. They are not applicable to our problem that considers systems with a discrete set of configurations. The state-feedback control technique for regulation problems developed in Aberkane et al. (2021); Goncalves et al. (2017, 2015) is moreover not applicable to our problem because it assumes a fixed timing set-point for the outputs. Refs (van den Boom et al. 2020; Alirezaei et al. 2012; Fusco et al. 2018; Brunsch et al. 2012; Nasri et al. 2014; Schafaschek et al. 2020; Aberkane et al. 2021) propose methods that are applicable to systems that, similar to our case, have a discrete set of configurations. However, none of these methods is applicable to systems with dynamic deadlines and uncontrollable dynamic workloads.

The solutions most closely related to our solution are those of Dirza et al. (2019); Menguy et al. (2000); Schafaschek et al. (2020); Lahaye et al. (2008); Silva and Maia (2016). The solution for the max-plus tracking problem of Dirza et al. (2019) is MPC, where the model prediction is essential to cope with the unpredictably changing output reference. The approach of Dirza et al. (2019) assumes a perfect model for the system's dynamics, and therefore no system feedback is needed. The technique does not apply to systems with uncertainties and dynamic workload variations. We integrate state feedback with MPC to cope with the combination of dynamic workloads and model uncertainties. For the max-plus tracking problems of Menguy et al. (2000); Schafaschek et al. (2020); Lahaye et al. (2008); Silva and Maia (2016), solutions for just-in-time control have been proposed. The synthesis for just-in-time control for systems with dynamic deadlines is done based on *residuation* theory. This theory is well suited to compute the latest activation times for the control inputs such that the outputs occur in time to meet their deadlines. However, just-in-time execution is not necessarily optimal in terms of implementation cost when there is a trade-off between the implementation cost and the execution time of a subsystem selected in response to a received workload. It is desirable to maintain some slack with respect to the dynamically changing deadlines to allow to optimize implementation cost for future workloads. Nevertheless, residuation theory is useful in our setting as well. We use a result from residuation theory in deriving feedback-gain matrices, at design time, allowing for just-in-time activation of the control inputs at run-time to ensure as-soon-as-possible execution of the selected subsystem.

The optimization and regulation techniques of van den Boom et al. (2020); Alirezaei et al. (2012); Fusco et al. (2018); Brunsch et al. (2012); Nasri et al. (2014); Schafaschek et al. (2020) for discrete control problems are applicable to our problem by considering the worst-case output time reference and taking the worst-case workload. The discrete-control tracking technique of Aberkane et al. (2021) is applicable to our problem by considering the worst-case workload. However, assuming worst cases would lead to conservative and expensive designs in terms of implementation cost. We may expect the result to be often far from optimal. The results may only be competitive when changes in the output-time reference and workload variation are small.

In conclusion, existing max-plus-based solutions in literature do not explicitly consider all ingredients of the problem addressed in this paper: tracking dynamic deadlines at a minimal implementation cost while we have uncontrollable dynamic workloads, (bounded) model uncertainties, and a limited discrete number of system configurations to respond to dynamic changes. This paper proposes MPC to cope with the unpredictable output reference and the unpredictable dynamic workloads. We combine it with a state-feedback controller for the timing of the configuration changes and to cope with the uncertainties in the system model. The feedback-gain matrices of the state-feedback controller are computed using residuation theory. The approach takes into account the time and cost of switching configurations.

3 Motivating example: an image-processing pipeline

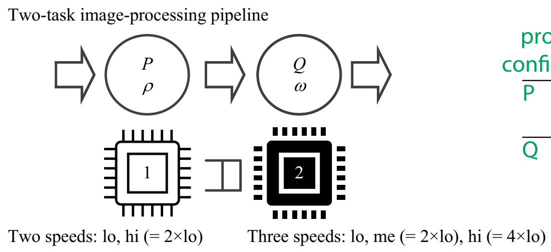
Consider a simplified two-processor image-processing pipeline as illustrated in Fig. 1. The pipeline iteratively processes received images in subsequent tasks P and Q. The processing workload for each of the received images depends on the content of the data (e.g., the number of features in the images) and cannot be controlled. The processor speeds (their operating frequencies) can be controlled. Processor 1 has two speeds, processor 2 has three. The nominal processing times ρ_n and ω_n of tasks P and Q for the workload/processor speed combinations are given in the table. Workloads capture coarse-grained variations in task processing times. For given workloads, processing times may still show fine-grained variations (because of image content, data caching, etc.). We assume that processing times vary within intervals $[\tau - 4, \tau + 2]$, with τ the nominal task processing times ρ_n or ω_n for any of the workload/processor speed combinations. The table in Fig. 1 also provides the power consumed per processor in each of its configurations. Power consumption is quadratically proportional to speed. Changing the operating frequency of a processor is fast compared to data processing. For the example, we assume it takes 1 time unit with a power consumption of 1 unit of power. The memory allocation allows to buffer the data of two images already processed by P but not yet by Q.

We can model the timing of the image-processing pipeline as an SMPLS. The system has 2 workloads and 6 ($= 2 \times 3$) configurations. This leads to 12 ($= 2 \times 6$) subsystems. These subsystems only differ in their timing. The SMPLS has three states x_1, x_2, and x_3. States $x_2(k)$ and $x_3(k)$ correspond to the completion times of execution $k - 1$ of tasks P and Q[1]. State $x_1(k)$ is the completion time of execution $k - 2$ of task Q. It captures the buffer capacity of 2 between tasks P and Q. Only if $Q(k-2)$ has completed, $P(k)$ can start. The SMPLS has two inputs u_1 and u_2 capturing the activation times of the processor configuration with which the received workload is processed. The SMPLS has one output y, capturing the production time of the output data.

As a basis for the SMPLS model (elaborated in Section 4), we derive equations that relate the completion times of executions k of tasks P and Q ($x_2(k + 1)$ and $x_3(k + 1)$), buffer availability time ($x_1(k + 1)$), and output ($y(k)$) to earlier task completion times ($x_2(k)$ and $x_3(k)$), earlier buffer availability ($x_1(k)$), and inputs ($u_1(k)$ and $u_2(k)$) that trigger reconfiguration (switching). We assume given initial states $x(0)$, capturing the initial availability of the processors and buffer space. For simplicity, we assume that switching time is always 1, even if processor speeds do not change. We obtain the following equations:

$$x_1(k + 1) = x_3(k) \tag{1}$$

[1] The indexing of the state variables was chosen to facilitate readability of the analyses later in the paper.


Two-task image-processing pipeline
P
ρ
Q
ω
1
2
Two speeds: lo, hi (= 2×lo)
Three speeds: lo, me (= 2×lo), hi (= 4×lo)

processor configuration		processing time	wl 1	wl 2	power
P	hi	ρ_n	15	30	80
	lo		30	60	20
Q	hi	ω_n	10	5	80
	me		20	10	20
	lo		40	20	5

Fig. 1 An image-processing pipeline

$$
\begin{aligned}
x_2(k+1) &= \max(x_2(k), x_1(k), u_1(k)+1) + \rho_n \\
x_3(k+1) &= \max(x_3(k), x_2(k+1), u_2(k)+1) + \omega_n \\
y(k) &= x_3(k+1)
\end{aligned}
$$

The example is kept small so that it can serve as running example. It captures all aspects relevant for the problem at hand though. It has a discrete number of controllable configurations with a switching cost, uncontrollable workloads, fine-grained variations in processing times, and an inverse relationship between processing times and implementation cost (power consumption). These characteristics are shared by many cyber-physical systems with some form of data-intensive time-critical control implemented on embedded compute platforms, in the medical domain (imaging equipment), in automotive (assisted/autonomous driving), and in manufacturing (chip lithography, production printing). Such systems may all benefit from the proposed approach.

4 Max-plus preliminaries

This section introduces notations and background on MPA and (S)MPLS.

4.1 Max-Plus Algebra (MPA)

The domain of MPA is the set of real numbers $\mathbb{R}$ plus ϵ, denoted $\mathbb{R}_\epsilon$. Element ϵ is referred to as minus infinity or *null*. The basic operations of MPA are maximization and addition, represented by $\oplus$ and $\otimes$:

$$
z_1 \oplus z_2 = \max(z_1, z_2)\ ,\ z_1 \otimes z_2 = z_1 + z_2\ ,\ z_1 \in \mathbb{R}_\epsilon\ ,\ z_2 \in \mathbb{R}_\epsilon\ , \tag{2}
$$

where we adopt the convention that for all $z_1 \in \mathbb{R}_\epsilon$,

$$
\max(z_1, \epsilon) = \max(\epsilon, z_1) = z_1\ ,\ z_1 + \epsilon = \epsilon + z_1 = \epsilon\ . \tag{3}
$$

The structure $(\mathbb{R}_\epsilon, \oplus, \otimes)$ is called MPA (Baccelli et al. 1992; Cuninghame-Green 1979). Operations $\oplus$ and $\otimes$ are respectively called the max-plus-algebraic addition and max-plus-algebraic multiplication, since many properties and concepts from linear algebra can be translated to MPA by replacing $+$ by $\oplus$ and $\times$ by $\otimes$ (Baccelli et al. 1992; Cuninghame-Green 1979).

Matrix operations can be defined in MPA as follows. Let $Z_1 \in \mathbb{R}_\epsilon^{m\times n}$, $Z_2 \in \mathbb{R}_\epsilon^{m\times n}$, and $Z_3 \in \mathbb{R}_\epsilon^{n\times p}$, for three natural numbers, m, n, and p.

$$\begin{aligned}\left(Z_1 \oplus Z_2\right)_{i,j} &:= \left(Z_1\right)_{i,j} \oplus \left(Z_2\right)_{i,j} = \max\left(\left(Z_1\right)_{i,j}, \left(Z_2\right)_{i,j}\right), \\ \left(Z_1 \otimes Z_3\right)_{i,j} &:= \bigoplus_{h=1}^{n}\left(\left(Z_1\right)_{i,h} \otimes \left(Z_3\right)_{h,j}\right) = \max_{h=1,\ldots,n}\left(\left(Z_1\right)_{i,h} + \left(Z_3\right)_{h,j}\right),\end{aligned} \tag{4}$$

where i and j are respectively the row-index and column-index of the matrix entries and $(.)_{i,j}$ indicates a matrix with its entries (Baccelli et al. 1992; Cuninghame-Green 1979).

4.2 MPLS

A max-plus-linear system (MPLS), where $k \in \mathbb{N}_0$ is the iteration variable, is a set of equations of the following form:

$$\begin{aligned} x(k+1) &= A \otimes x(k) \oplus B \otimes u(k), \\ y(k) &= C \otimes x(k) \oplus D \otimes u(k). \end{aligned} \tag{5}$$

In Eq. 5, $x(k)$, $u(k)$, and $y(k)$ are respectively the state, the input, and the output vectors at the k^{th} iteration of the system; A, B, C, and D are the system matrices. The elements of the vectors are *event times* (Baccelli et al. 1992; Cuninghame-Green 1979).

Motivating example: Returning to the example of Section 3, the event times of interest are task completion times, buffer availability times, output times, and reconfiguration triggers. We can derive a matrix representation from the equations in Eq. 1 that follows the format of Eq. 5. The following is obtained by appropriate substitution for $x_2(k+1)$ in the left-hand side of the third equation and distribution of the processing times over the max expressions:

$$\left[\begin{array}{c:c} A & B \\ \hdashline C & D \end{array}\right] = \left[\begin{array}{ccc:cc} \epsilon & \epsilon & 0 & \epsilon & \epsilon \\ \rho_n & \rho_n & \epsilon & \rho_n + 1 & \epsilon \\ \rho_n + \omega_n & \rho_n + \omega_n & \omega_n & \rho_n + \omega_n + 1 & \omega_n + 1 \\ \hdashline \rho_n + \omega_n & \rho_n + \omega_n & \omega_n & \rho_n + \omega_n + 1 & \omega_n + 1 \end{array}\right]. \tag{6}$$

For concrete values of ρ_n and ω_n, this provides an MPLS model for a given workload running in a specific system configuration.

In line with our motivating example, we assume MPLS with *a single output*. The numbers of states and inputs are given by N_x and N_u; states and inputs are indexed by $S_x = \{1, 2, ..., N_x\}$ and $S_u = \{1, 2, ..., N_u\}$.

4.3 SMPL systems

Recall that we consider systems with (uncontrollable) workload variations and (controllable) system configurations. Such systems can be modeled as a switched MPLS (SMPLS) that consists of subsystems that each are an MPLS (see Van Den Boom and De Schutter (2004)). We consider single-output SMPLS in the form of Eq. 7 where matrices $A^{(i)} \in \mathbb{R}_\epsilon^{N_x\times N_x}$, $B^{(i)} \in \mathbb{R}_\epsilon^{N_x\times N_u}$, $C^{(i)} \in \mathbb{R}_\epsilon^{1\times N_x}$, and $D^{(i)} \in \mathbb{R}_\epsilon^{1\times N_u}$ are the system matrices for the i^{th} subsystem, ranging over $S_{sub} = \{1, 2, ..., N_{sub}\}$.

$$\begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix} = \begin{bmatrix} A^{(i)}(k) & B^{(i)}(k) \\ C^{(i)}(k) & D^{(i)}(k) \end{bmatrix} \otimes \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}. \tag{7}$$

5 Problem setting

In this section, Section 5.3 in particular, we precisely define the scheduling and control problem addressed in this paper. Section 5.1 first introduces the SMPLS specification we take as a starting point. Section 5.2 defines the optimization objectives, specifically deadline misses and implementation cost. Section 5.4 illustrates the scheduling and control challenge through the motivating example.

5.1 SMPLS specification

We assume that the system of interest is specified in the form of a *single-output* SMPLS as introduced in Section 4.3, with *measurable states* and with the *output defined by one of the states* (i.e., there is a state $j \in S_x$ such that for all subsystems $i \in S_{sub}$, $C^{(i)} = A^{(i)}_j$ and $D^{(i)} = B^{(i)}_j$, where, for any matrix M, M_j denotes row j of M). This is a valid assumption if one of the system tasks produces the output and the system states correspond to task completion times. Since the system matrices model delays, we further assume that *all non-ϵ entries are non-negative real numbers*. Moreover, we assume that the model has *no redundancy* (i.e., there are no two states $j_1, j_2 \in S_x$, $j_1 \neq j_2$, such that for all subsystems $i \in S_{sub}$, $A^{(i)}_{j_1} = A^{(i)}_{j_2}$ and $B^{(i)}_{j_1} = B^{(i)}_{j_2}$). Finally, we assume *structural finiteness*. That is, all states and outputs after a system iteration are affected by either one of the inputs or one of the system states at the iteration start. Since we consider single-output SMPLS with the output equal to one of the states, this condition holds if and only if for all subsystems $i \in S_{sub}$, the matrix $[A^{(i)}\ B^{(i)}]$ does not have a null row. Such a null row would imply that the respective state is continuously ϵ, which is not meaningful.

Motivating example: For a multiprocessor image-processing pipeline, it is reasonable to assume that task completion times and buffer availability (the system states) are measurable. The equations of Eq. 1 and the system matrices in Eq. 6 show that our motivating example satisfies the other mentioned assumptions. Structural finiteness follows because all system states (task completion times and buffer availability) depend on completion of earlier task executions (actually implying that the $A^{(i)}$ matrices have no null row).

Let $WS = \{1, 2, ..., N_{ws}\}$ be the set of *workloads*. Since the system configuration is controllable upon receipt of an uncontrollable workload, WS partitions S_{sub} into subsystem sets $S^{(w)}_{sub}$ corresponding to workloads $w \in WS$. If workload w is received, the system can only respond with a subsystem in $S^{(w)}_{sub}$.

Motivating example: As explained in Section 3, our motivating example has 12 subsystems for 2 workloads. The workloads $WS = \{1, 2\}$ partition the subsystems in two sets of 6 subsystems each: $S_{sub} = S^{(1)}_{sub} \cup S^{(2)}_{sub}$. A system configuration is defined by the received workload and the speed of the two processors, e.g., (2, hi, lo) is the configuration in which workload 2 is received and processed with processor 1 at high speed and processor 2 at low speed. The two sets of subsystems are then $S^{(1)}_{sub} = \{1 = (1, \text{hi}, \text{hi}), 2 = (1, \text{hi}, \text{me}), 3 = (1, \text{hi}, \text{lo}), 4 = (1, \text{lo}, \text{hi}), 5 = (1, \text{lo}, \text{me}), 6 = (1, \text{lo}, \text{lo})\}$ and $S^{(2)}_{sub} = \{7 = (2, \text{hi}, \text{hi}), 8 = (2, \text{hi}, \text{me}), 9 = (2, \text{hi}, \text{lo}), 10 = (2, \text{lo}, \text{hi}), 11 = (2, \text{lo}, \text{me}), 12 = (2, \text{lo}, \text{lo})\}$. The system matrices $A^{(i)}$, $B^{(i)}$, $C^{(i)}$, and $D^{(i)}$, for $i \in S_{sub}$, all follow the structure of Eq. 6, with the nominal values ρ_n and ω_n given in Fig. 1.

Our approach takes into account the time needed to switch between configurations. Switching causes a delay in the activation time of the upcoming subsystem. The switching time when switching from subsystem q to subsystem p of an SMPLS can be expressed as a vector

$d^{(q,p)} = [d_1^{(q,p)}\ d_2^{(q,p)}\ \ldots\ d_{N_u}^{(q,p)}]^T$ or, equivalently, as a diagonal matix $D_{sw}^{(q,p)} \in \mathbb{R}_\epsilon^{N_u \times N_u}$ with the $d_j^{(q,p)}$, $1 \leq j \leq N_u$, values on the diagonal. Switching from subsystem q to subsystem p can then be taken into account in the state-space equations as follows:

$$\begin{aligned}\begin{bmatrix} x(k+1) \\ y(k) \end{bmatrix} &= \begin{bmatrix} A^{(p)} & B^{(p)} \\ C^{(p)} & D^{(p)} \end{bmatrix} \otimes \begin{bmatrix} x(k) \\ u(k) + d^{(q,p)} \end{bmatrix} \\ &= \begin{bmatrix} A^{(p)} & B^{(p)} \otimes D_{sw}^{(q,p)} \\ C^{(p)} & D^{(p)} \otimes D_{sw}^{(q,p)} \end{bmatrix} \otimes \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}. \end{aligned} \tag{8}$$

Motivating example: Recall the equations in Eq. 1 specifying our motivating example. These equations include the (constant) switching time of 1 as a delay after the activation times $u(k)$, in line with the first equality in Eq. 8. The system matrices in Eq. 6 include the switching time conform the second equality in Eq. 8.

In the remainder, we assume that the *switching time is included in the system matrices* $B^{(i)}$ and $D^{(i)}$ of the subsystem $i \in S_{sub}$ to which the SMPLS is switching. When switching times vary with to and from configurations, this leads to a potentially quadratic increase in the number of subsystems in the SMPL model, if arbitrary switching is allowed. If switching time does not depend on to and from configurations, the number of subsystems is unaffected.

We finally assume *bounded additive uncertainty* in the system matrices of the SMPLS specification. That is, $A^{(i)}(k)$, $B^{(i)}(k)$, $C^{(i)}(k)$, and $D^{(i)}(k)$ represent the *uncertain* system matrices of the i^{th} subsystem at system iteration k, such that, for given error bounds $\alpha_{lb}, \alpha_{ub} \in \mathbb{R}_0^+$, every matrix element m of any of these matrices falls within interval $[m - \alpha_{lb}, m + \alpha_{ub}]$.

Motivating example: As explained in Section 3, workload processing times may vary within given bounds. From the model given in Eq. 6, it then easily follows that the uncertainty in the system matrices is bounded and additive.

5.2 Dynamic deadlines and performance objectives

The first performance objective that needs to be optimized is the number of deadline misses. We assume a dynamically varying reference of inter-arrival times of outputs, $\Delta_{ref} : \mathbb{R}_0^+ \to \mathbb{R}^+$, given as a piece-wise constant function over the time domain. The reference production time of the output is defined through a signal y_{ref}, with $y_{ref}(k)$ for system iteration k given by

$$y_{ref}(k) = \begin{cases} \Delta_{ref}(0), & k = 0\,, \\ y_{ref}(k-1) + \Delta_{ref}(y_{ref}(k-1)), & k \geq 1\,. \end{cases} \tag{9}$$

A deadline miss at iteration k, denoted $DM(k)$, for $k \geq 0$, is then defined as follows, where $y(k)$ is the actual output production time at iteration k:

$$DM(k) = \begin{cases} 1, & y(k) > y_{ref}(k)\,, \\ 0, & y(k) \leq y_{ref}(k)\,. \end{cases} \tag{10}$$

We define an evaluation window over which the performance objectives are analyzed, with an initial transient that is ignored. The number of missed deadlines $N_{miss}^{es,ew}(k)$ at iteration k, with $es \in \mathbb{N}$ the starting iteration for evaluation and $ew \in \mathbb{N}$ the length of the evaluation window, is defined as:

$$N_{miss}^{es,ew}(k) = \begin{cases} 0, & k < es + ew\,, \\ \sum_{l=0}^{ew-1} DM(k-l), & k \geq es + ew\,. \end{cases} \tag{11}$$

The second performance objective is cost of implementation, which should be minimized. To compute cost, we need the sequence of executed subsystems $\sigma(k) \in S_{sub}$ that specifies the subsystem at iteration k of the system. For a given es and ew, the implementation cost is defined as follows:

$$Cost^{es,ew}(k) = \begin{cases} 0, & k < es + ew\,, \\ \sum_{l=0}^{ew-1} Cost(k-l), & k \geq es + ew\,, \end{cases} \tag{12}$$

where $Cost(k)$, for $k \geq 1$, is given by a function $G : S_{sub}^2 \times \mathbb{R}_\epsilon^{N_x} \times \mathbb{R}_\epsilon \times (\mathbb{R}_\epsilon^{N_u})^2 \to \mathbb{R}_0^+$ that defines for each pair of subsystems $i, j \in S_{sub}$ the cost of executing j after i, given the state $x(k)$, the reference output deadline $y_{ref}(k)$, the activation times of j, $(u(k-1))$ and the completion times of j $(u(k))$. Note that $Cost(0)$, the cost of the first iteration, is left undefined by assuming that G depends on a previous iteration. This is a technicality, because we assume that G takes the switching cost into account, which is not defined for iteration 0. If needed, $Cost(0)$ can be defined by assuming a 0 switching cost, or by including a start-up cost. We do not do so, because we are primarily interested in steady-state behavior, and not in start-up behavior.

Motivating example: A concrete example of implementation cost is energy consumption. Let $P_j : S_{sub} \to \mathbb{R}_0^+$ be a function that gives the power dissipation of processor $j \in \{1, 2\}$, for running subsystem $\sigma \in S_{sub}$. The power numbers were already given in Section 3, Fig. 1. We further assumed that switching a processor configuration takes one time unit and comes with a power cost of one. The energy cost $E(k)$ in system iteration k then consists of a switching cost of 2, 1 unit of energy per processor, and an execution cost for both processors obtained by subtracting the switching times of 1 and the activation times $u(k-1)$ of configuration $\sigma(k)$ from the completion times of $\sigma(k)$, $u(k)$:

$$E(k) = 2 + [P_1(\sigma(k))\ P_2(\sigma(k))](u(k) - u(k-1) - 1) \tag{13}$$

The right-hand side of Eq. 13 follows the format of G introduced above, although the cost does in this case not depend on the previous configuration $\sigma(k-1)$ nor on the output reference. Moreover, the defined energy cost does not depend on whether or not a processor is actively used. This could be adapted, for example, to distinguish between a running and an idling processor.

5.3 Problem definition

We can now define our scheduling and control problem, namely how to minimize deadline misses and implementation cost for an SMPLS receiving an uncontrollable, dynamic sequence of workloads $w_k \in WS$ $(k \in \mathbb{N}_0)$ with dynamic deadlines $y_{ref}(k)$. The system has a discrete number of response options to respond to workloads from WS specified by subsystems $S_{sub}^{(w)}(w \in WS)$ such that $S_{sub}^{(w)} \cap S_{sub}^{(w')} = \emptyset$ $(w, w' \in WS, w \neq w')$ and

$S_{sub} = \bigcup_{w \in WS} S_{sub}^{(w)}$. Our (multi-objective) optimization problem is then as follows:

$$\begin{aligned} \min_{\sigma(k) \in S_{sub}^{(w_k)},\, u(k) \in \mathbb{R}_\epsilon^{N_u}} \quad & N_{miss}^{es,ew}(k),\ Cost^{es,ew}(k) \\ \text{s.t.} \quad x(k+1) &= A^{(\sigma(k))} \otimes x(k) \oplus B^{(\sigma(k))} \otimes u(k) \\ y(k) &= C^{(\sigma(k))} \otimes x(k) \oplus D^{(\sigma(k))} \otimes u(k), \end{aligned} \tag{14}$$

where $A^{(\sigma(k))}$, $B^{(\sigma(k))}$, $C^{(\sigma(k))}$, and $D^{(\sigma(k))}$ are the uncertain system matrices of the selected subsystem $\sigma(k)$ in iteration k of the system.

5.4 Motivating example

Figure 2 shows a Gantt chart of seven iterations of our motivating example. The horizontal axis is a time line. From bottom to top, the solid colored lines in the first seven rows show the evolution of the states, control inputs, output, and output reference, in line with the constraints in the problem definition of Eq. 14. The output $y(k)$ aligns with the availability of $x_3(k+1)$ in line with Eqs. 6 and 7. The top two rows of the Gantt chart show the pipelined execution of tasks P and Q. In line with Eq. 1, the start time of $P(k)$ is the maximum of $x_1(k)$ (buffer availability), $x_2(k)$ (completion of $P(k-1)$), and $u_1(k)+1$ (reconfiguration time plus switching delay, the latter indicated with the dashed colored lines); the start time of $Q(k)$ is the maximum of $x_3(k)$, $x_2(k+1)$ signalling the completion of $P(k)$ – the pipelining, and $u_2(k)+1$. Task completion times result from adding the (uncertain) execution times ρ and ω (shown in the blocks) of the executed subsystem. These task completion times are the state availability times for the next iteration. Interestingly, we may observe that, up to iteration 67, task Q is not critical for the timing of the system. In iteration 67 though, Q is executed at a low speed, to save energy, leading to a deadline miss ($y(67)$ being greater than $y_{ref}(67)$) and the need for a recovery action that increases the speed of processor 1 again in iterations 68 and 69.

Figure 3 shows further details on the same seven iterations of the running example. It shows the received workloads, the dynamic output reference, the two control inputs (switching times), the subsystems being executed, the number of missed deadlines (with $es = 10$, $ew =$

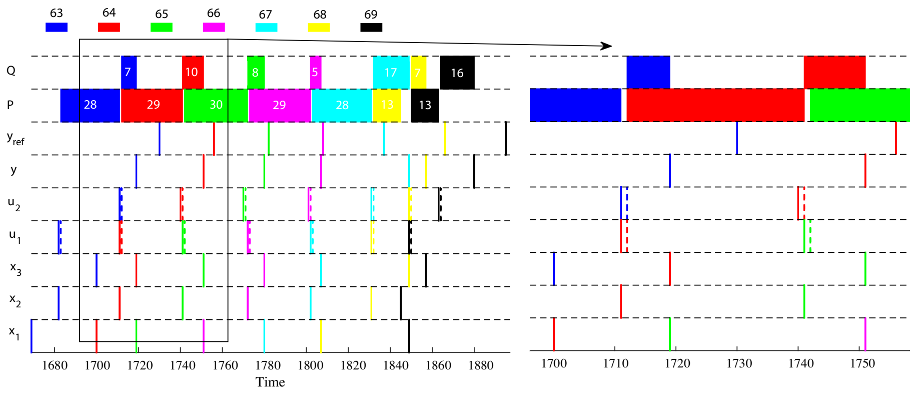


Fig. 2 Gantt chart of an execution fragment of the motivating example

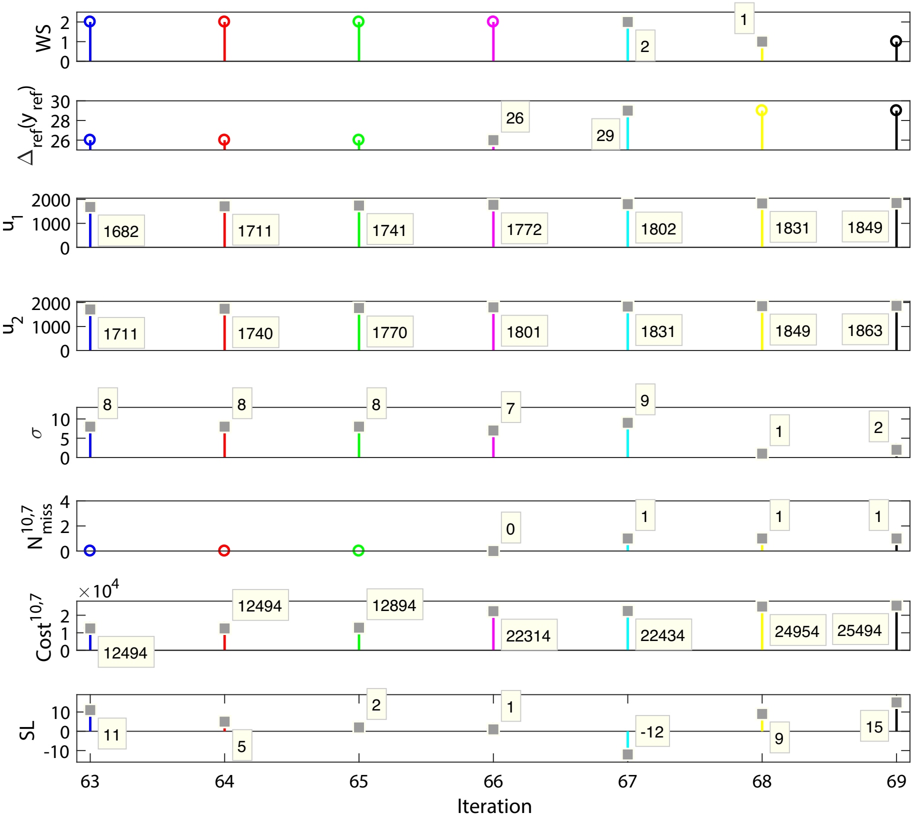


Fig. 3 Workloads, output reference, control inputs, executed subsystems, missed deadlines, cost, and slack of the execution fragment

7), the implementation cost, and the slack $y_{ref}(k) - y(k)$. As an example, following Eq. 13, the implementation (energy) cost for iteration 67 is as follows:

$$E(67) = 2 + [20\ 20][(1802 - 1772 - 1)\ (1831 - 1801 - 1)]^T = 1162\,. \tag{15}$$

Following Eq. 12, $Cost^{es,ew}$, with $ew = 7$, provides a sliding window of aggregated energy cost over seven iterations. Figure 3 shows this aggregated cost.

The system misses its deadline at iteration 67 (indicated by the negative slack) because it executes subsystem 9 with a low cost (1162, see Eq. 15) in an attempt to save energy. The next deadlines are met again as higher-speed and higher-cost subsystems are selected. The number of deadline misses in the evaluation window of seven iterations $N_{miss}^{10,7}$ stays 1.

The execution snapshot shown in Figs. 2 and 3 illustrates the combined challenge of selecting appropriate subsystems σ and switching times u in response to uncontrollable workloads trading off deadline misses and implementation cost in an attempt to minimize both.

6 Tracking dynamic output inter-arrival times

This section proposes an integrated scheduling and control method to solve the problem defined in Section 5.3. Section 6.1 gives an overview of the approach. Section 6.2 makes the assumptions explicit under which our method is applicable. Section 6.3 presents a state-feedback control approach to schedule subsystem switches. Section 6.4 describes computation of the feedback-gain matrices. Section 6.5 provides an MPC approach to select the appropriate subsystem at run-time in response to a received workload. Section 6.6 discusses the controller implementation.

6.1 Overview of the proposed approach

Figure 4 illustrates the overall control structure. The *integrated controller* has two parts, a state-feedback control and a model-predictive control. It simultaneously decides the timing of switching $u(k)$ and the subsystem $\sigma(k)$ executed for each system iteration k. Given a subsystem to execute, the state-feedback controller determines the earliest possible starting time for this subsystem, given up-to-date information on the (uncertain) system states. Without uncertainty in the execution times of the system, state feedback would not be needed. The MPC decides on the subsystem to execute in response to a received workload by looking ahead a given number of iterations, the *prediction horizon*. It uses a weighted sum of the (normalized) objectives, deadline misses and implementation cost, to select the best subsystem. It takes into account the state-feedback control in its predictions. The controller uses a *nominal and worst-case subsystems* for both the MPC prediction and the computation of the $u(k)$. Subsystems are actuated as late as possible (ALAP, just in time) to enable as-soon-as-possible (ASAP) execution of the subsystems through a *feedback gain*, using a specific *feedback-gain matrix* per subsystem. The γ block in Fig. 4 represents the backward shift

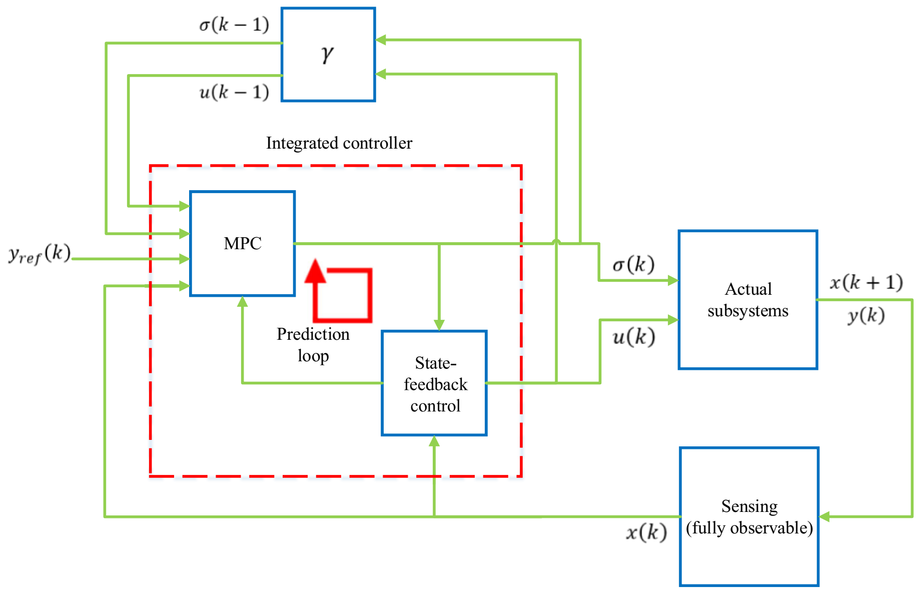


Fig. 4 The proposed feedback control structure

operator (see Baccelli et al. (1992)), which is conceptually similar to the Z-transformation in discrete-time control system theories.

6.2 Structural properties of our SMPLS

Problem definition Eq. 14 starts from a single-output SMPLS specification with bounded model uncertainties and the subsystems partitioned into sets corresponding to configurations that may be chosen to respond to uncontrollable workloads. For our approach to work, we need a few assumptions.

Assumption 1 - *Inputs affect states*: The input matrices should not have null columns: $\forall i \in S_{sub}\, \forall j_1 \in S_u\, \exists j_2 \in S_x : B^{(i)}_{j_2,j_1} \neq \epsilon$.

Assumption 2 - *Controllability*: A state is *uncontrollable* in a subsystem when it is not directly affected by inputs (see Prou and Wagneur (1999)). So a state is uncontrollable if the corresponding row in the subsystem's input matrix is the null row. Let $S^{(i)}_{un}$ be the (indices corresponding to) uncontrollable states in subsystem $i \in S_{sub}$, defined as $S^{(i)}_{un} = \{j_1 \in S_x \mid \forall j_2 \in S_u : B^{(i)}_{j_1,j_2} = \epsilon\}$. We assume that for all subsystems $i_1, i_2 \in S_{sub}$, $S^{(i_1)}_{un} = S^{(i_2)}_{un}$. That is, all subsystems have the same uncontrollable states, henceforth referred to as the uncontrollable states of the SMPLS, denoted S_{un}. The set $S_{co} = S_x \setminus S_{un}$ is the set of *controllable states*. We assume that the state defining the output is controllable.

Assumption 3 - *Full actuation*: We assume a fully actuated system. Since we assume a system model without redundancy, a subsystem $i \in S_{sub}$ is *fully actuated* when the number of controllable states in the subsystem is the same as the number of inputs. Because all subsystems have the same controllable states and the same inputs, all subsystems, and hence the SMPLS, are fully actuated if $|S_{co}| = |S_u| = N_u$.

Motivating example: From the model in Eq. 6, it is clear that all inputs affect at least one state (Assumption 1). This naturally follows from the fact that control inputs reconfigure processors before a task executes and hence influence the task completion times. State 1 of the example system, corresponding to buffer availability, is uncontrollable in all subsystems; buffer availability is typically only indirectly controlled through task executions. The other two states, corresponding to task completion times, are controllable in all subsystems. Hence, Assumption 2 is satisfied, with $S_{un} = \{1\}$ and $S_{co} = \{2, 3\}$. As a result, also Assumption 3 is satisfied. The example SMPLS is fully actuated.

6.3 Scheduling and coping with uncertainty: state-feedback control

The system matrices have bounded uncertainty due to small unpredictable deviations in subsystem execution times (Section 5.1). Since we assume that states are measurable, these deviations can be taken into account through state feedback. The state-feedback control approach presented in this subsection controls the switching between subsystems by deciding the timing of the control inputs.

Control law: Using state feedback $x(k)$, the input activation times for subsystem $i \in S_{sub}$ executed in the current iteration are determined as follows:

$$u(k) = K^{(i)} \otimes x(k) \ , \tag{16}$$

where $K^{(i)}$ is the feedback-gain matrix of subsystem i.

Closed-loop system dynamics: To obtain the closed-loop system dynamics, we eliminate the input signals $u(k)$ from the max-plus equations of Eq. 7 using the control law of Eq. 16.

We arrive at the following, where $A_{cl}^{(i)}$ and $C_{cl}^{(i)}$ are the state and output matrices of the closed-loop system, respectively:

$$x(k+1) = A_{cl}^{(i)} \otimes x(k), \text{ with } A_{cl}^{(i)} = A^{(i)} \oplus \left(B^{(i)} \otimes K^{(i)}\right) , \tag{17}$$

$$y(k) = C_{cl}^{(i)} \otimes x(k), \text{ with } C_{cl}^{(i)} = C^{(i)} \oplus \left(D^{(i)} \otimes K^{(i)}\right) . \tag{18}$$

The feedback-gain matrix $K^{(i)}$ must satisfy both Eqs. 17 and 18. From the assumption that the single output equals one of the system states (Section 5.1), we can conclude that Eq. 17 implies Eq. 18. Hence, it suffices to consider Eq. 17.

We divide matrices $A_{cl}^{(i)}$, $A^{(i)}$, and $B^{(i)}$ into controllable and uncontrollable parts. Null rows in the $B^{(i)}$ matrices do not provide meaningful information for computing the feedback-gain matrices. Let $A_{co}^{(i)}$, $B_{co}^{(i)}$, and $A_{cl,co}^{(i)}$ be the $|S_{co}|$ rows of $A^{(i)}$, $B^{(i)}$, and $A_{cl}^{(i)}$ corresponding to the controllable parts. We then obtain the following equation for the controllable states, as the starting point for the computation of the feedback-gain matrices:

$$[A_{cl,co}^{(i)}]_{|S_{co}| \times N_x} = [A_{co}^{(i)}]_{|S_{co}| \times N_x} \oplus \left([B_{co}^{(i)}]_{|S_{co}| \times N_u} \otimes [K^{(i)}]_{N_u \times N_x}\right). \tag{19}$$

6.4 Computing feedback-gain matrices

Starting from Eq. 19, we consider two cases for computing the feedback gain for the i^{th} subsystem in the closed-loop system:

$$B_{co}^{(i)} \otimes K^{(i)} \prec A_{co}^{(i)} , \tag{20}$$
$$B_{co}^{(i)} \otimes K^{(i)} \succeq A_{co}^{(i)} . \tag{21}$$

Note that the subsystem to be executed after i can only be started after completing i, i.e., at $A_{co}^{(i)} \otimes x$, in line with Eq. 21. Inequality Eq. 20 has a trivial solution, namely minus-infinity matrices for all $i \in S_{sub}$. It does not provide meaningful information for computing the feedback-gain matrices.

Solving Eq. 21: With gains based on Eq. 21, we potentially delay the execution of subsystems. This allows to potentially cover scheduling policies from ASAP execution up to and including ALAP or just-in-time execution with respect to the current deadline. To solve Eq. 21, we again consider two cases, distinguishing whether or not the equality version of Eq. 21 has at least one solution.

Solving the equality version of Eq. 21: In this case, we try to solve equality

$$B_{co}^{(i)} \otimes K_{eq}^{(i)} = A_{co}^{(i)} , \tag{22}$$

with $K_{eq}^{(i)}$ being the solution. Equation 22 may not have a unique solution. To solve Eq. 22, we apply methods of Butkovic (2010); Tsiamis and Maragos (2019) that build on residuation theory. We aim to find the greatest solution. Taking the greatest solution as the basis for the feedback-gain matrices provides ALAP actuation of the control inputs.

Solvability of Eq. 22: The number of unknown elements of the feedback-gain matrices is $N_u \times N_x$. Given the dimensions of $A_{co}^{(i)}$ in Eq. 22, we have $N_{co} \times N_x$ max-plus equations. Since $|S_{co}| = N_u$ (Assumption 3 in Section 6.2), we have sufficient equations to solve Eq. 22.

Solving Eq. 22: We find $K_{eq}^{(i)}$ solving Eq. 22 in two steps. In Step 1, based on the infinite and finite entries of $A_{co}^{(i)}$, we separate the equations into two parts. Then, Step 2 solves the equations in two sub-steps; first, we determine the infinite entries of the matrix $K_{eq}^{(i)}$ after which we derive its finite entries.

Step 1: The equations from Eq. 22 can be re-ordered separating infinite and finite entries of $A^{(i)}_{co}$. Let $N_{\epsilon A}$ and $N_{fA} = N_{co} \times N_x - N_{\epsilon A}$ be the number of infinite and finite entries of $A^{(i)}_{co}$. Equation 22 can then be written as follows:

$$[B^{(i)}_{co,\epsilon A}]_{N_{\epsilon A} \times (N_u \times N_x)} \otimes [K^{(i)}_{eq,v}]_{(N_u \times N_x) \times 1} = [\epsilon]_{N_{\epsilon A} \times 1} \,, \tag{23}$$

$$[B^{(i)}_{co,fA}]_{N_{fA} \times (N_u \times N_x)} \otimes [K^{(i)}_{eq,v}]_{(N_u \times N_x) \times 1} = [A^{(i)}_{co,v}]_{N_{fA} \times 1} \,, \tag{24}$$

where $K^{(i)}_{eq,v}$ is $K^{(i)}_{eq}$ in vector format, $A^{(i)}_{co,v}$ contains all finite entries of $A^{(i)}_{co}$ in vector format, and $B^{(i)}_{co,\epsilon A}$ resp. $B^{(i)}_{co,fA}$ are appropriately derived from $B^{(i)}_{co}$.

Step 2a, computing infinite $K^{(i)}_{eq,v}$ entries: Equation 23 provides a collection of max-plus additions between $B^{(i)}_{co,\epsilon A}$ and $K^{(i)}_{eq,v}$ entries that all have to resolve to ϵ. For finite $B^{(i)}_{co,\epsilon A}$ entries, the $K^{(i)}_{eq,v}$ entry must be ϵ. Infinite $B^{(i)}_{co,\epsilon A}$ entries do not provide information about the $K^{(i)}_{eq,v}$ entry. We then proceed to Step 2b.

Solvability of Eq. 24*, a doubly* $\mathbb{R}$*-astic matrix*: By construction, all $A^{(i)}_{co,v}$ entries are finite. It follows that Eq. 24 does not have a solution if $B^{(i)}_{co,fA}$ contains a null row. Similarly, if $B^{(i)}_{co,fA}$ has a null column with index j, then we can set $(K^{(i)}_{eq,v})_{j,1}$ to any value in a solution for Eq. 24. Assumption 1 in Section 6.2 implies that $B^{(i)}_{co,fA}$ has no null columns. Since $B^{(i)}_{co,fA}$ refers to the controllable states, it also has no null rows. That is, it is *doubly* $\mathbb{R}$*-astic*. This is a necessary (but not sufficient) condition for Eq. 24 to have a unique greatest solution.

Step 2b, computing finite $K^{(i)}_{eq,v}$ entries: Following (Butkovic 2010; Tsiamis and Maragos 2019), Eq. 24 has a solution if and only if

$$B^{(i)}_{co,fA} \otimes \underbrace{\Big(\big(-(B^{(i)}_{co,fA})^T \big) \otimes' A^{(i)}_{co,v} \Big)}_{*} = A^{(i)}_{co,v} \,. \tag{25}$$

In Eq. 25, $(.)^T$ provides the conjugate matrix in max-plus context and the $\otimes'$ operation is defined as follows:

$$\Big(\big(-(B^{(i)}_{co,fA})^T \big) \otimes' A^{(i)}_{co,v} \Big)_{j,1} := \min_h \Big(\big(-(B^{(i)}_{co,fA})^T \big)_{j,h} + (A^{(i)}_{co,v})_{h,1} \Big) \,, \tag{26}$$

which is a matrix-vector multiplication in *min-plus algebra* (Heidergott et al. 2014).

The * marked expression is a solution for Eq. 24 derived using residuation theory (Baccelli et al. 1992; Cuninghame-Green 1979), named the *principal solution* in Butkovic (2010); Tsiamis and Maragos (2019). If it is a solution, it is the greatest solution (with respect to the canonical order on vectors in max-plus algebra). This provides a solution for the finite entries of $K^{(i)}_{eq,v}$.

A solution for Eq. 22: By combining the finite and infinite entries computed for $K^{(i)}_{eq,v}$ in Steps 2b and 2a, we obtain solution $K^{(i)}_{eq}$ for Eq. 22.

Causality: An important practical concern for implementing a controller is the *causality* of the control law. A causal control law avoids the need to activate control inputs before a running subsystem has completed. Our feedback-gain matrices must be *causal* to ensure causality of the control law. A matrix is causal if all its finite entries are non-negative (see Goncalves et al. (2015)).

A solution for Eq. 21*, the* ***feedback-gain matrix***: The solution $K^{(i)}_{eq}$ computed for Eq. 22 is not necessarily causal. Note though that we do not necessarily need a solution for equality Eq. 22. We need a solution for *in*equality Eq. 21. So $K^{(i)}_{eq}$ can safely be made causal by replacing finite negative entries with 0. The resulting matrix $K_{ca}^{(i)}$ provides a solution for inequality Eq. 21. This solution is our feedback-gain matrix for subsystem $i \in S_{sub}$. The greater a gain matrix, the later the subsystem is executed. Taking the principal (greatest) solution for Eq. 24 and then turning this into the smallest causal solution for Eq. 21 provides ALAP actuation of the control inputs for ASAP execution of the subsystem.

Motivating example: Following the above steps for subsystem 8 of our running example, we derive the following solutions for Eqs. 22 and 21, resp.:

$$K_{eq}^{(8)} = \begin{bmatrix} -1 & -1 & \epsilon \\ 29 & 29 & -1 \end{bmatrix}, K_{ca}^{(8)} = \begin{bmatrix} 0 & 0 & \epsilon \\ 29 & 29 & 0 \end{bmatrix}. \tag{27}$$

$K_{ca}^{(8)}$ is the feedback-gain matrix for subsystem 8. Given that the nominal execution time of task P in configuration 8 (= (2, hi, me)) is 30 (see Fig. 1), we may interpret this gain matrix as an attempt to execute the next subsystem as soon as possible (the 0 entries) while actuating the second processor as late as possible, hiding the reconfiguration delay of 1 (the 29 entries).

Solving the strict inequality version of Eq. 21: In this case, i.e., when Eq. 22 does not have a solution, we need to find a lower-bound matrix $[A_{colb}^{(i)}]_{N_{co}\times N_x} \in \mathbb{R}_{\epsilon,0}^{+}$ for subsystem i, sufficiently greater than $A_{co}^{(i)}$, that guarantees a solution for Eq. 22 with $A_{colb}^{(i)}$ substituted for $A_{co}^{(i)}$. Since $B_{co}^{(i)}$ is doubly $\mathbb{R}$-astic, we can always find this matrix. Following the reasoning of the first case, $[A_{colb,v}^{(i)}]_{(N_{co}\times N_x)\times 1}$ can be found by solving Eq. 25, where $A_{colb,v}^{(i)}$ is the vector format of the unknown matrix $A_{colb}^{(i)}$. Using Eqs. 25 and 26, the following conditions have to be met:

$$\left(A_{colb,v}^{(i)}\right)_{j_1,1} - \max_{j_2}\left(\left(B_{co,fA}^{(i)}\right)_{j_1,j_2}, \min_{j_3}\left(\left(-\left(B_{co,fA}^{(i)}\right)^T\right)_{j_2,j_3} + \left(A_{colb,v}^{(i)}\right)_{j_3,1}\right)\right) = 0,$$

$$\left(A_{colb,v}^{(i)}\right)_{j_1,1} - \max\left(\left(A_{colb,v}^{(i)}\right)_{j_1,1}, \left(A_{co,v}^{(i)}\right)_{j_1,1}\right) = 0. \tag{28}$$

Equation 28 can be solved numerically using the Levenberg-Marquardt algorithm (Moré 1978). Given the assumptions on our SMPLS, it may be possible to solve them more efficiently, but this is left as future work. Since the feedback-gain matrices are computed at design time, maximal efficiency is not essential.

A solution for Eq. 21*, the feedback-gain matrix*: From this point onward, we can obtain the feedback-gain matrix for the considered subsystem as in the first case, by turning the solution for Eq. 25 obtained with $A_{colb,v}^{(i)}$ into a causal matrix providing the desired solution for Eq. 21.

The gain matrices obtained through the outlined approach realize ASAP execution of the selected subsystem, with ALAP activation of the control inputs. ASAP execution is meaningful when the primary goal is to avoid deadline misses. ASAP execution builds up slack with respect to the unpredictable dynamically changing deadlines, reducing the risk of missing future deadlines and providing room for optimizing future implementation cost. It may be interesting to explore other control activation and subsystem scheduling policies in future work. Care has to be taken that the run-time selection of subsystems remains tractable. The MPC in our approach (see Fig. 4), developed in the next subsection, looks ahead in time to select the best possible subsystem to execute in response to a workload and a given output time reference, given the selected ASAP scheduling policy with ALAP control activation.

6.5 Selecting the subsystem to execute: model-predictive control

We introduce an MPC approach to select a subsystem to execute in response to a received workload. The MPC looks ahead for a set number of iterations, the optimization window, optimizing a weighted sum of normalized deadline misses and normalized implementation cost. Looking ahead improves performance and provides robustness against the model uncertainties, uncontrollable dynamic workload variations, and unpredictably changing deadlines.

Predicting output times: We assume a finite prediction horizon. The workload to be processed in a given iteration of the system is known; workloads for any later iterations are unknown. The worst-case workload scenario is taken into account for any such future iterations. As an example, assuming a prediction horizon of two, Eq. 29 below shows how to predict the output production times at iteration k, where $i \in S_{sub}$ is the subsystem executed at iteration k and matrices with the '(wc)' superscript are matrices of the subsystem executed in response to the worst-case workload.

$$\begin{bmatrix} y(k) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} C^{(i)} \\ C^{(wc)} \otimes A^{(i)} \end{bmatrix} \otimes x(k) \oplus \begin{bmatrix} D^{(i)} & \epsilon \\ C^{(wc)} \otimes B^{(i)} & D^{(wc)} \end{bmatrix} \otimes \begin{bmatrix} u(k) \\ u(k+1) \end{bmatrix}. \tag{29}$$

Equation 29 can be adapted to larger prediction horizons. Besides worst-case C and D matrices, also worst-case A and B matrices are then needed.

To realize an MPC approach with a look-ahead of at least two, we need to identify the worst-case workload w_{wc} and we need to define subsystems $S_{sub}^{(wc)}$ to respond to it. We describe three different ways to do so.

First, a natural worst-case workload exists: A well-defined worst-case workload exists if there is a subsystem that has matrices that are at least the matrices of any of the other subsystems and that can only be used in response to a single workload. That particular workload is then the worst-case workload. In our MPC technique, we can then use the subsystems allowed in response to that workload (as defined in Section 5.1).

Motivating example: The SMPLS for our running example does not have a natural worst-case workload. All the subsystems follow the model of Eq. 6. We see, for instance, that $A_{21} = \rho_n$ and $A_{33} = \omega_n$. The values given in Fig. 1 for these parameters show that $\omega_n > \rho_n$ for the first workload, whereas $\rho_n > \omega_n$ for the second workload. In other words, for one workload, task P is the slowest, while for the other workload task Q is the slowest.

If there is no clear worst-case workload, we can construct a synthetic worst-case workload with its possible response(s) for use in the MPC. Note that the worst-case workload w_{wc} is essentially only an index. So without natural worst-case workload, we can simply take $w_{wc} = N_{ws} + 1$. The challenge is to define meaningful subsystem responses to it. We explain two possible ways.

Second, use application knowledge: The system model represents a system at hand that may provide information that allows to construct a worst-case workload and its subsystem responses. For the motivating example, the combined worst-case task execution times may represent a worst-case workload. These worst-case task execution times can be used in the matrices of Eq. 6 to create a worst-case (slowest) subsystem response. Since the subsystem responses for the workload scenarios in the example are essentially scaled versions of the slowest response to a particular workload, we may construct additional subsystem responses for the worst-case workload by applying the same scaling.

Motivating example: Fig. 5 (right table) lists six subsystems as possible responses to a synthetic worst-case workload taking the worst-case task execution times (left table). The

processor configuration		processing time	wl w_{wc}
P	hi	ρ_n	30
	lo		60
Q	hi	ω_n	10
	me		20
	lo		40

subsystem	processor configurations	
13	hi	hi
14	hi	md
15	hi	lo
16	lo	hi
17	lo	md
18	lo	lo

Fig. 5 The worst-case workload and its responses for the running example

slowest response (subsystem 18) results from operating both processors at low speed. Its matrices are obtained by filling in the ρ_n and ω_n values of Fig. 5 in the model of Eq. 6:

$$\left[\begin{array}{c|c} A^{(18)} & B^{(18)} \\ \hline C^{(18)} & D^{(18)} \end{array}\right] = \left[\begin{array}{ccc|cc} \epsilon & \epsilon & 0 & \epsilon & \epsilon \\ 60 & 60 & \epsilon & 61 & \epsilon \\ 100 & 100 & 40 & 101 & 41 \\ \hline 100 & 100 & 40 & 101 & 41 \end{array}\right]. \tag{30}$$

The system matrices for the other five subsystems in Fig. 5 can be obtained by scaling the task execution times given in the figure by factors 2 and 4, respectively, depending on the processor speeds in the configuration. The six subsystems complement the twelve subsystems already given in Section 5.1.

Third, use worst-case system responses: A general way to obtain a worst-case subsystem response for a synthetic worst-case workload is to take the element-wise maximum over the subsystem matrices of all possible subsystems.

Motivating example: Taking the element-wise maximum over the matrices of all twelve subsystems identified in Section 5.1 gives the following matrices:

$$\left[\begin{array}{c|c} A^{(13)} & B^{(13)} \\ \hline C^{(13)} & D^{(13)} \end{array}\right] = \left[\begin{array}{ccc|cc} \epsilon & \epsilon & 0 & \epsilon & \epsilon \\ 60 & 60 & \epsilon & 61 & \epsilon \\ 80 & 80 & 40 & 81 & 41 \\ \hline 80 & 80 & 40 & 81 & 41 \end{array}\right]. \tag{31}$$

The result is less conservative than the result of Eq. 30. The subsystem is indexed 13, as before complementing the twelve actual subsystems of the system.

Feedback-gain matrices: With the worst-case workload and its subsystem responses defined, we compute feedback-gain matrices for these subsystems, to the extent not already done, following the procedure given in Section 6.4.

Predicting deadline misses: As explained in Section 5.2, our optimization targets two objectives. The first objective is the number of deadline misses over a given evaluation window. Ideally, the MPC should consider the full evaluation window in its predictions. But this may be computationally infeasible. Hence, we introduce a separate optimization window length $ow \in \mathbb{N}$, defining the MPC prediction horizon. This parameter provides a trade-off between optimality and computation time. Let $\hat{\sigma}(1) \in S_{sub}^{(w_k)}$ be a subsystem that can be executed in response to a received workload $w_k \in WS$; let $\hat{\sigma}(2), \ldots, \hat{\sigma}(ow) \in S_{sub}^{(wc)}$ be subsystem responses for the worst-case workload. Eq. 32 defines the number of deadline misses over the prediction horizon when executing subsystem sequence $\hat{\sigma}$ in response to the k^{th} workload received w_k. The $\lambda_j \in \mathbb{R}_0^+$, for $1 \leq j \leq ow$, are weighting factors to trade off the importance of a missed deadline at iteration k and future deadline misses. Note that DM in Eq. 32 depends (implicitly) on the executed subsystems from $\hat{\sigma}$.

$$N_{miss}^{ow,\hat{\sigma}}(k) = \sum_{l=0}^{ow-1} \lambda_{ow-l} \times DM(k+l)\,. \tag{32}$$

In our experiments, λ_j is set as follows:

$$\lambda_1 = 1\,,\ \lambda_2 = 2\,,\ \lambda_j = \lambda_{j-1} + \lambda_{j-2}\,,\ \forall\, j \geq 3\,. \tag{33}$$

This is conceptually similar to the Fibonacci sequence (see Sigler (2003)). This gives more weight to deadline misses in the near future than to later deadline misses (similar to, but not as strongly as, exponentially increasing λ_j). This choice is reasonable because we have workload knowledge for iteration k, whereas we use conservative workloads and execution delays for future iterations.

To meaningfully compare different objectives in MPC, we have to properly normalize objectives. The maximum value for $N_{miss}^{ow,\hat{\sigma}}(k)$ over a window of length ow is $\sum_{j=1}^{ow} \lambda_j$ and its minimum value is 0. Therefore, the normalized version of $N_{miss}^{ow,\hat{\sigma}}(k)$ is defined as follows:

$$N_{miss,no}^{ow,\hat{\sigma}}(k) = \frac{1}{\sum_{j=1}^{ow} \lambda_j} \times N_{miss}^{ow,\hat{\sigma}}(k)\,. \tag{34}$$

Predicting implementation cost: The second optimization objective defined in Section 5.2 is the implementation cost. The implementation cost is also computed over an iteration window of length ow. The predicted cost for executing subsystems $\hat{\sigma}$ as defined above in response to workload w_k in iteration k is:

$$Cost^{ow,\hat{\sigma}}(k) = \sum_{l=0}^{ow-1} \lambda_{ow-l} \times Cost(k+l)\,, \tag{35}$$

where $Cost(k)$ captures the cost of an iteration, depending among others on the executed subsystem, as elaborated in Section 5.2. The cost prediction is weighted using the same λ_j parameters as the prediction of deadline misses.

If a synthetic worst-case workload was defined for the the deadline miss prediction, then we need to have a meaningful implementation cost for the execution of subsystems defined in response to this worst-case workload as well. We need both the execution and switching cost for the newly introduced subsystems. One option is to assume constant values, for instance, the values obtained by taking the maximum over the implementation cost of all actual subsystems and by assuming the maximum switching cost for all switches. When through scaling multiple subsystem responses were defined, the implementation cost may be scaled accordingly for those subsystems.

We need to normalize cost to meaningfully compare cost with deadline misses. Note that the MPC needs to consider $|S_{sub}^{(w_k)}| \times |S_{sub}^{(wc)}|^{(ow-1)}$ possible sequences for received workload w_k. Let $S(k)$ be the set of all these sequences. The cost for subsystem sequence $\hat{\sigma}$ at iteration k is normalized considering the minimum and maximum costs over all sequences in $S(k)$:

$$Cost_{no}^{ow,\hat{\sigma}}(k) = \frac{Cost^{ow,\hat{\sigma}}(k) - \min_{\hat{\sigma}' \in S(k)} Cost^{ow,\hat{\sigma}'}(k)}{\max_{\hat{\sigma}' \in S(k)} Cost^{ow,\hat{\sigma}'}(k) - \min_{\hat{\sigma}' \in S(k)} Cost^{ow,\hat{\sigma}'}(k)}\,. \tag{36}$$

Optimization algorithm: The MPC uses a weighting factor $\lambda_{dl} \in [0, 1]$ to trade off the implementation cost and the deadline misses over the optimization window. We assume that the reference output times, the deadlines, are given over the optimization window. We may assume worst-case output inter-arrival times or a continuation of the latest inter-arrival time if these reference deadlines are not given. Since the number of deadline misses and implementation cost are never negative, our MPC uses a linear cost function, instead of a quadratic one. Recall from above that $S(k)$ is the set of all possible subsystem sequences over the prediction horizon to be considered by the MPC in response to the received workload at iteration k. The optimization problem solved by the MPC in each system iteration is as follows:

$$\begin{aligned} \min_{\hat{\sigma}\in S(k)} \quad & \lambda_{dl} \times N_{miss,no}^{ow,\hat{\sigma}}(k) + (1-\lambda_{dl}) \times Cost_{no}^{ow,\hat{\sigma}}(k) \\ \text{s.t.} \quad & u(k) = K^{(\hat{\sigma}(l))} \otimes x(k) \\ & x(k+1) = \left(A^{(\hat{\sigma}(l))} \oplus \left(B^{(\hat{\sigma}(l))} \otimes K^{(\hat{\sigma}(l))}\right)\right) \otimes x(k) \\ & y(k) = \left(C^{(\hat{\sigma}(l))} \oplus \left(D^{(\hat{\sigma}(l))} \otimes K^{(\hat{\sigma}(l))}\right)\right) \otimes x(k)\,, \end{aligned} \tag{37}$$

where the $A^{(\hat{\sigma}(l))}$, $B^{(\hat{\sigma}(l))}$, $C^{(\hat{\sigma}(l))}$, $D^{(\hat{\sigma}(l))}$, and $K^{(\hat{\sigma}(l))}$ are the (nominal) system and feedback-gain matrices of the subsystems $\hat{\sigma}(l)$, for $1 \leq l \leq ow$.

Optimizing Eq. 37 returns the optimal sequence $\hat{\sigma} \in S(k)$ at iteration k. The first element of this sequence, $\hat{\sigma}(1)$, is selected as the executed subsystem $\sigma(k)$. It is actuated at times $u(k)$. The MPC computes the $u(k)$ (using the feedback gains) and $y(k)$ over the prediction horizon to compute the deadline misses and implementation cost for its cost function. It uses the closed-loop model as the closed-loop matrices can be computed offline. Output $y(k)$ is not explicitly computed because it aligns with one of the controllable states.

Computational complexity: The MPC explores all $|S_{sub}^{(w_k)}| \times |S_{sub}^{(wc)}|^{(ow-1)}$ options in $S(k)$. Per option, per iteration within the prediction horizon, two matrix multiplications (for $u(k)$ and $x(k+1)$) are needed. The first one takes $N_x \times N_u$ binary additions and $N_u \times (N_x - 1)$ binary max operations; the second one takes N_x^2 binary additions and $N_x \times (N_x - 1)$ binary max operations. The number of matrix multiplications can be further reduced by avoiding repetitive computations (for instance, occurring for larger prediction horizons in combination with a single worst-case subsystem response).

Motivating example: Figs. 2 and 3 discussed in Section 5.4 are obtained by applying the MPC for the running example with one step look-ahead, i.e., $ow = 1$, and weighting factor $\lambda_{dl} = 0.8$ in the cost function. We refer to this controller as C_{mpc1}. Figure 6 shows the execution of the same seven iterations as in Fig. 3, but with the subsystems selected by the MPC with two steps look-ahead, i.e., $ow = 2$, and using the worst-case workload and its six possible responses of Fig. 5 in its prediction. The controller thus considers 36 predictions to determine the subsystem to execute at iteration k. It assumes that the reference output inter-arrival time $\Delta_{ref}(y_{ref}(k))$ is the same as the reference at iteration $k-1$, i.e., $\Delta_{ref}(y_{ref}(k-1))$. We refer to this controller as $C_{mpc2,wc6}$. Comparing Fig. 6 with Fig. 3 shows that looking ahead for more than one step is beneficial to reduce the risk of missing an output deadline. Interestingly, we also pay a smaller energy cost (15434) (see $Cost^{10,7}(69)$ in Fig. 6) compared to the case with one step look-ahead (25494) (see $Cost^{10,7}(69)$ in Fig. 3).

Figure 7 shows the results with the subsystems selected by MPC with two steps look-ahead and the single worst-case subsystem response of Eq. 31. The resulting controller $C_{mpc2,wc1}$ determines the optimal response from six predictions. Comparing Fig. 7 with Figs. 3 and 6 confirms that looking ahead for more than one step is beneficial for reducing deadline misses. However, with $C_{mpc2,wc1}$, we pay a greater energy cost (31394) (see $Cost^{10,7}(69)$ in Fig. 7) compared to the energy cost of $C_{mpc2,wc6}$ (15434) (see $Cost^{10,7}(69)$ in Fig. 6).

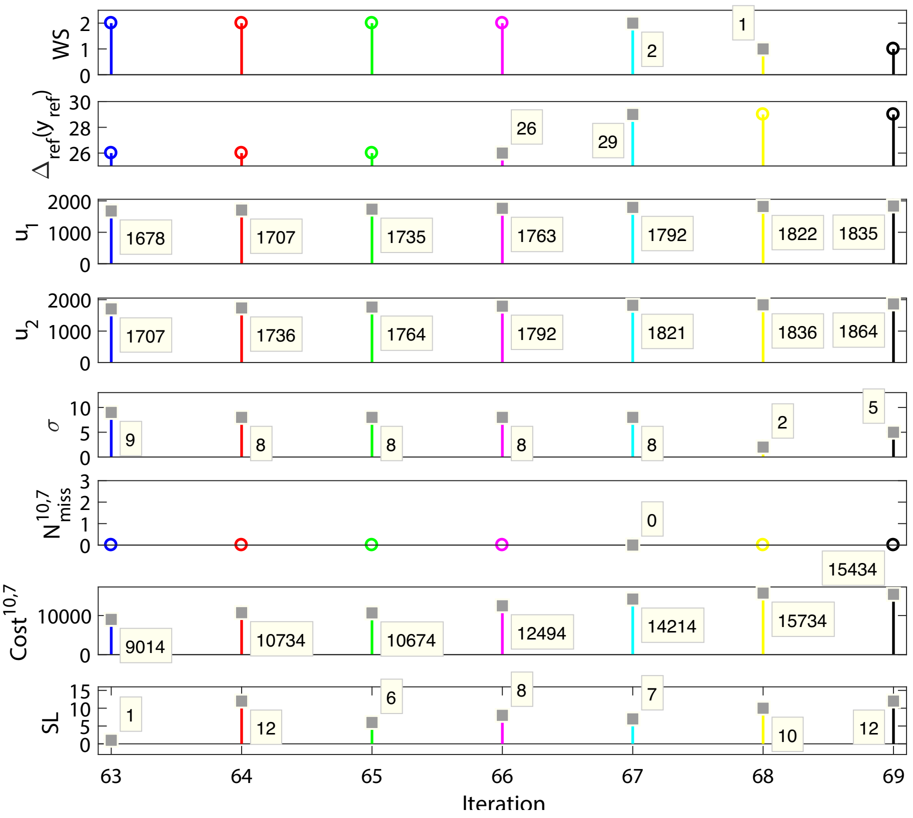


Fig. 6 Workloads, output reference, control inputs, executed subsystems, missed deadlines, cost, and slack for $C_{mpc2,wc6}$

6.6 Controller implementation

We implemented our control approach in MATLAB. In each system iteration, the subsystem selected by the MPC is executed, initiating the reconfiguration (subsystem switching) at the actuation times computed through the control law (Eq. 16). Actuation times are computed as soon as the relevant states are available. For instance, considering Fig. 2, this means that $u_1(k)$ is computed as soon as $x_1(k)$ and $x_2(k)$ are available; $u_2(k)$ is then computed as soon as also $x_3(k)$ is available. Following the ASAP execution policy, reconfiguration is started earlier than prescribed by the control law if all relevant states (task completions, buffer availability) have been observed. For example, processor 2 running task Q can be reconfigured as soon as both the previous instance of Q and the current instance of P have completed. Reconfiguration is delayed if (the relevant part of) the preceding system iteration has not yet completed. For instance, reconfiguration of a processor cannot start before the previous task execution on that processor has completed (captured by states x_2 and x_3 for the two processors). Our implementation neglects the time needed to compute the controller outputs. In practice, this time can be considered part of the reconfiguration (switching) times included in the B and D matrices.

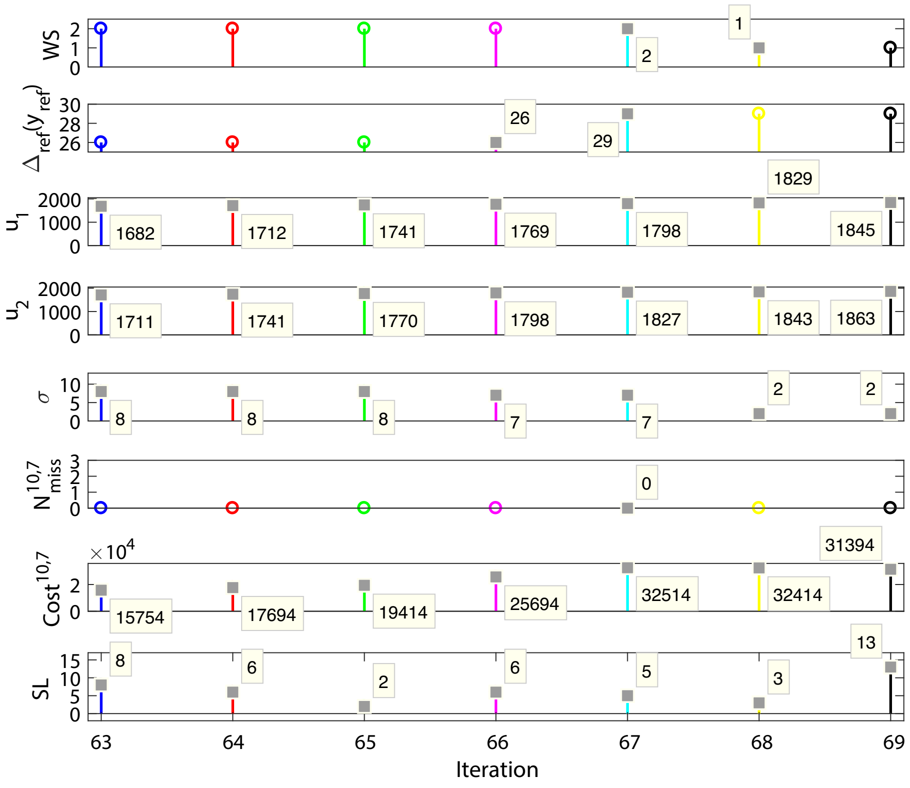


Fig. 7 Workloads, output reference, control inputs, executed subsystems, missed deadlines, cost, and slack for $C_{mpc2,wc1}$

7 Simulation setup and results

We evaluate our approach on the motivating example. We provide the simulation setup and present results that show the effectiveness of our approach.

7.1 Simulation setup

We implemented our approach in MATLAB (version R2017a) using the MPA toolbox (Stanczyk 2016). Experiments were done on a computer with a 2.40 GHz CPU with 12 GB of RAM and a 64-bit operating system.

Distribution of workload scenarios: We choose a discrete uniform distribution of workloads, with occurrence probability 0.5 for both WS_1 and WS_2.

Task-execution time distribution: We choose a single discrete probability distribution function (PDF) that captures the uncertainty in the execution times of the two tasks in the various configurations; see Fig. 8 (left). Recall that task processing times are assumed to vary within the bounded interval $[\tau - 4, \tau + 2]$, with τ being the nominal task processing times ρ_n or ω_n for any combination of the workloads and processor configurations. That is, task execution

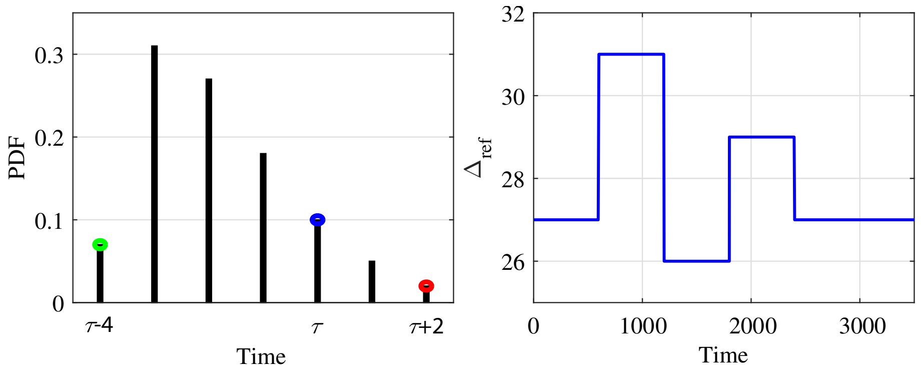


Fig. 8 PDF for task execution times (left) and output reference (right)

times vary within a range of seven time units. The execution times for a system iteration are then obtained by adding the lower-bound value for the task execution time corresponding to the subsystem considered for execution in that iteration to the value sampled from the uncertainty distribution. The green, blue, and red circles correspond to the lower bound, the nominal value, and the upper bound of the task execution times.

Dynamic output reference: The reference for the output inter-arrival times is set to a piece-wise constant function over the time domain; Fig. 8 (right).

Reference solutions for comparison: We consider two reference solutions to evaluate our MPC approach. The first one runs the processors at their fastest speeds (and hence the highest cost). It executes subsystem 1 when workload 1 is received and subsystem 7 for workload 2. This prioritizes avoiding deadline misses and follows approaches from van den Boom et al. (2020); Alirezaei et al. (2012); Fusco et al. (2018); Brunsch et al. (2012); Nasri et al. (2014); Schafaschek et al. (2020) as discussed in Section 2. The second reference solution runs the processors at their lowest speeds (i.e., at the lowest cost). It executes subsystems 6 and 12 for workloads 1 and 2. This reference solution provides an approximate lower bound on the implementation cost. We refer to these two reference controllers as C_{hc} and C_{lc}, respectively.

We perform two sets of experiments to assess the performance of our approach, one verifying robustness against model uncertainties and one verifying robustness against workload variations. We compare the five controllers discussed so far, i.e., C_{mpc1}, $C_{mpc2,wc1}$, $C_{mpc2,wc6}$, C_{hc}, and C_{lc}. The system runs 110 iterations in each sub-experiment. By setting the starting iteration for evaluation to 10, i.e., $es = 10$, we cut ten initial iterations (numbered 0 to 9) to avoid transition effects. The length of the evaluation window is set to 100.

Verifying robustness against model uncertainties: For these experiments, we generate three different sequences of 110 workload scenarios. To derive sufficient data for verifying the robustness of the controllers against the model uncertainties, for each of these three workload sequences, each controller runs fifty times with different task execution times. As a result, for each controller, we obtain 150 values for the number of deadline misses and energy cost.

Verifying robustness against workload variations: For these experiments, we generate 50 different sequences of 110 workload scenarios. For each of these 50 workload sequences, each controller is run once. This gives fifty different values for the number of deadline misses and energy cost per controller.

7.2 Simulation results

Figures 9 and 10 provide the results for the two sets of experiments. The horizontal axes show the number of sub-experiments, 150 for the first set and 50 for the second set. The top rows in the graphs show $N_{miss}^{10,100}(110)$, the number of deadline misses over the iterations, excluding 10 initial iterations. The bottom rows provide $Cost^{10,100}(110)$, the energy cost over the iterations, again excluding 10 initial iterations. To compare the controllers, Tables 1 and 2 show the averages for the two objectives. We may draw the following conclusions:

– Comparing the green C_{mpc1} controller with the black $C_{mpc2,wc1}$ controller and the pink $C_{mpc2,wc6}$ controller shows that looking ahead for more than one step reduces the risk of missing output deadlines, in line with earlier observations made in Section 6.5.
– Comparing $C_{mpc2,wc1}$ (black) and C_{mpc1} (green) shows that looking ahead two iterations with a single worst-case response (given in Eq. 31) increases conservatism. The risk of missing deadlines is reduced but cost increases (because of higher processor speeds in selected subsystem responses).
– Comparing $C_{mpc2,wc1}$ (black) and $C_{mpc2,wc6}$ (pink) shows that with six subsystems to respond to the worst-case workload both the deadline misses and the energy cost are reduced. On the one hand, looking ahead for more than one step reduces deadline misses. By assuming the worst case in the look-ahead iterations, the controller typically chooses a faster configuration. This naturally needs a higher energy cost for execution. On the other hand, the six subsystems potentially used in response to the synthetic worst-case workload add flexibility to use a suitable slower response when there is enough time slack. A slower configuration has a lower energy cost for execution. Moreover, since the

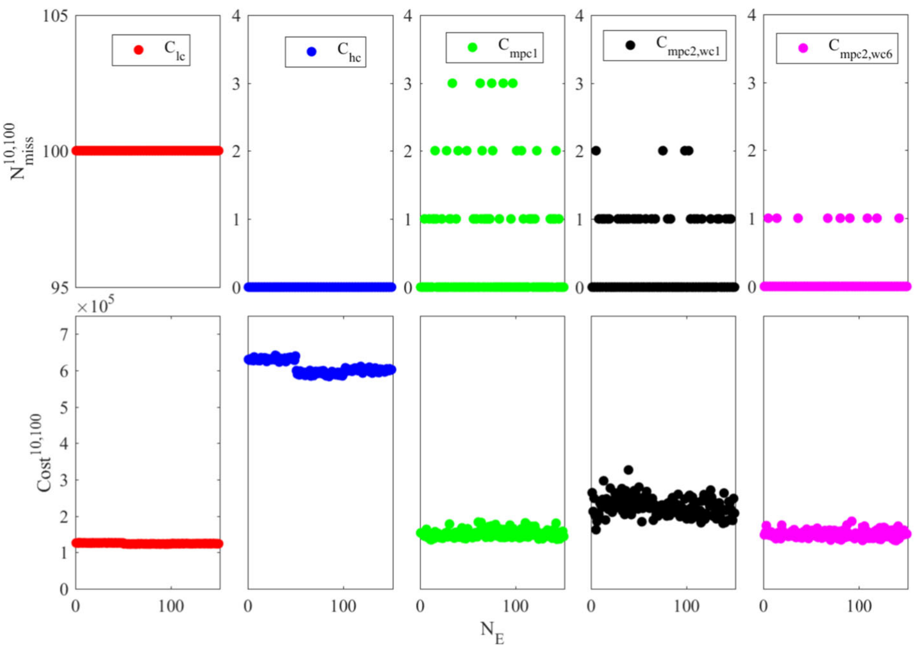


Fig. 9 Performance analysis of the designed controllers based on the experiments done for verifying robustness against model uncertainties

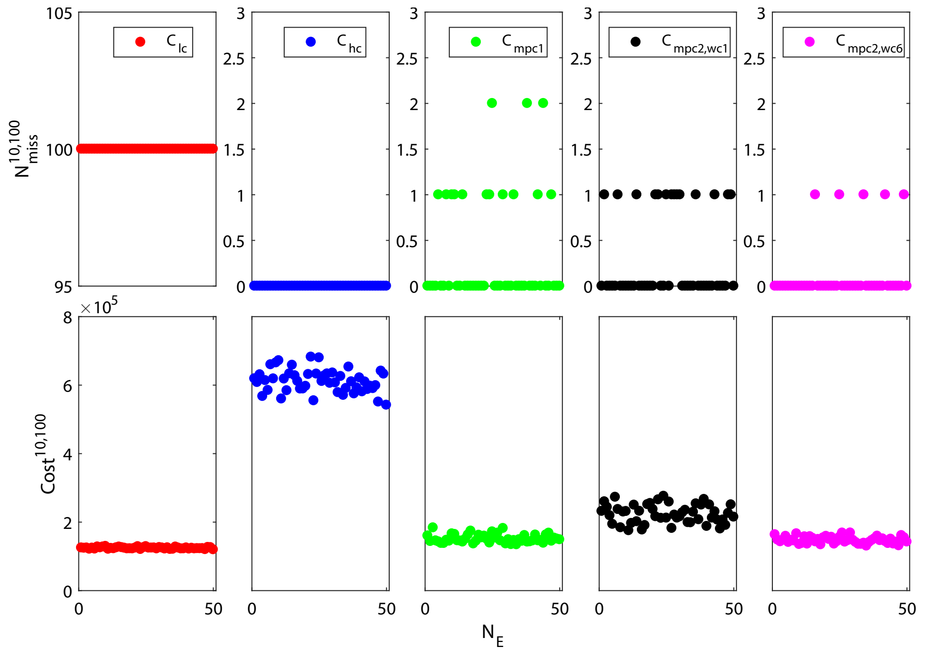


Fig. 10 Performance analysis of the designed controllers based on the experiments done for verifying robustness against workload variations

six worst-case-workload responses come from the system model, prediction accuracy is improved.

- Controllers C_{mpc1} (green), $C_{mpc2,wc1}$ (black), and $C_{mpc2,wc6}$ (pink) take, on average over 200 experiments per controller, 0.0009s, 0.0052s, and 0.0293s to select a subsystem and compute the control inputs. As expected, the prediction horizon and the number of subsystems possible in response to the worst-case workload directly impact the running times.
- From the observations so far, we conclude that both $C_{mpc2,wc6}$ (pink) and $C_{mpc2,wc1}$ (black) are robust against workload variations and model uncertainties, in the sense that the number of deadline misses is small, with $C_{mpc2,wc6}$ outperforming $C_{mpc2,wc1}$. Over the 200 experiments per controller, the controllers do not miss more than two deadlines in any run of 100 system iterations and their energy cost does not show outliers. The results show that robustness can be enhanced by increasing look ahead and defining suitable worst-case subsystem responses for use in predictions. If the time needed for predictions is too high because of the prediction window length in combination with the number of

Table 1 Performance summary for model-uncertainty experiments

	C_{lc}	C_{hc}	C_{mpc1}	$C_{mpc2,wc1}$	$C_{mpc2,wc6}$
$N_{miss}^{10,100}(110)$	100	0	0.4467	0.2600	0.0600
$Cost^{10,100}(110)$	124440	608090	151370	227090	150260
			↑21.6%, ↓75.1%	↑82.5%, ↓62.6%	↑20.7%, ↓75.3%

Table 2 Performance summary for workload experiments

	C_{lc}	C_{hc}	C_{mpc1}	$C_{mpc2,wc1}$	$C_{mpc2,wc6}$
$N_{miss}^{10,100}(110)$	100	0	0.3400	0.2800	0.1000
$Cost^{10,100}(110)$	124010	610994	151390	223200	147330
			$\uparrow$22.1%, $\downarrow$75.2%	$\uparrow$80.0%, $\downarrow$63.5%	$\uparrow$18.8%, $\downarrow$75.9%

worst-case system responses, parameter λ_{dl} in the MPC cost function may be adapted to prioritize avoiding deadline misses. Also the nominal system matrices of subsystems can be adapted. Choosing nominal system matrices closer to the upper-bound matrices increases conservatism in the responses without adding computation time to the MPC predictions. Note that upper and lower bounds on execution delays are typically given and cannot be (easily) controlled. But the nominal system model can be chosen freely within the bounds.

– Comparing $C_{mpc2,wc6}$ (pink) with the C_{lc} controller (red) shows a higher energy cost, of around 20% (see Tables 1 and 2), for $C_{mpc2,wc6}$. Recall that C_{lc} minimizes energy cost, without considering deadline misses. This leads to 100% deadline misses. We conclude that the energy cost of our MPC controller $C_{mpc2,wc6}$ is acceptable (and to a large part unavoidable) if one wants to avoid deadline misses.
– Comparing $C_{mpc2,wc6}$ (pink) with the (fastest, highest-cost) C_{hc} controller (blue) shows that the pink $C_{mpc2,wc6}$ controller misses some output deadlines, whereas the blue C_{hc} controller does not. However, $C_{mpc2,wc6}$ saves about 75% in energy cost compared to C_{hc}. If the number of deadline misses of a controller designed following our MPC approach is unacceptable in comparison to the fastest reference controller, its robustness may be further tuned as discussed above.

8 Conclusion and future work

We presented a control method to track a dynamic reference on the output times of a system that has a discrete number of system configurations to respond to uncontrollable received workloads. By integrating state-feedback and predictive control, we obtain controllers that minimize deadline misses and implementation costs. The controllers are robust against both uncontrollable workload variations and model uncertainties. Our approach takes into account the switching time and cost. Considering a two-core reconfigurable multi-processor system as a motivating example and using MATLAB simulations, we demonstrated the effectiveness of our method.

In the current setup, our approach provides ASAP execution of system responses with ALAP actuation. Realizing ALAP execution or other scheduling schemes is an interesting topic for future work. Another next step may be to lift the assumption on the measurability of system states by adding a max-plus-based observer to the framework. The requirement that the SMPLS is fully actuated may also be a limiting factor. It is interesting to see whether our approach can be adapted to under- and/or over-actuated systems.

Acknowledgements This research was supported in part by the Electronic Components and Systems for European Leadership (ECSEL) Joint Undertaking under grant number H2020-ECSEL-2017-2-783162 through the FitOptiVis project (Sau et al. 2021). We thank the anonymous reviewers, whose comments have helped us greatly to improve our method and its presentation in the paper.

Declarations

The authors declare that they have no financial or personal interests that influenced the work reported in this paper.

References

Aberkane S, Kara R, Amari S (2021) Modelling and feedback control for a class of Petri nets with shared resources subject to strict time constraints using max-plus algebra. Int J Syst Sci 52(14):1–16

Alirezaei M, Van Den Boom TJ, Babuska R (2012) Max-plus algebra for optimal scheduling of multiple sheets in a printer, American Control Conference (ACC). Montreal, QC, Canada, pp 1973–1978

Aström KJ, Hägglund T (2006) PID control. IEEE Control Syst Mag 1066

Baccelli F, Cohen G, Olsder GJ, Quadrat JP (1992) Synchronization and Linearity. Wiley, New York

Bajaber F, Elshawi R, Batarfi O, Altalhi A, Barnawi A, Sakr S (2016) Big data 2.0 processing systems: Taxonomy and open challenges. J Grid Comput 14(3):379–405

Basten T et al (2020) Scenarios in the design of flexible manufacturing systems. System-Scenario-based Design Principles and Applications, pp 181–224, Springer

Brunsch T, Raisch J, Hardouin L (2012) Modeling and control of high-throughput screening systems. Control Eng Pract 20:14–23

Butkovic P (2010) Max-Linear Systems: Theory and Algorithms. Springer

Camacho EF, Alba CB (2013) Model Predictive Control. Springer

Chakraborty I, Mehta SS, Curtis JW, Dixon WE (2016) Compensating for time-varying input and state delays inherent to image-based control systems, American Control Conference (ACC). Boston, MA, USA, pp 78–83

Chen CLP, Zhang C (2014) Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Inf Sci 275:314–347

Cuninghame-Green RA (1979) Minimax algebra, vol 166. lecture notes in economics and mathematical systems. Springer-Verlag, Berlin

Dirza R, Marquez-Ruiz A, Özkan L, Mendez-Blanco CS (2019) Integration of max-plus-linear scheduling and control. Comput Aided Chem Eng 46:1279–1284

Dorf RC, Bishop RH (2008) Modern Control Systems. Pearson Prentice Hall

Doyle JC, Francis AB, Tannenbaum AR (2013) Feedback Control Theory. Courier Corporation

Fahad A, Alshatri N, Tari Z, Alamri A, Khalil I, Zomaya AY, Foufou S, Bouras A (2014) A survey of clustering algorithms for big data: Taxonomy and empirical analysis. IEEE Trans Emerg Top Comput 2(3):267–279

Fusco M, Semsar-Kazerooni E, Zegers JC, Ploeg J (2018) Decision making for connected and automated vehicles: A max-plus approach. Proc. IEEE 88^{th} Vehicular Technology Conference (VTC), Chicago, USA, pp 1–5

Geilen MCW et al (2020) Scenarios in dataflow modeling and analysis. In: System-Scenario-based Design Principles and Applications, pp 145–180, Springer

Gheorghita SV et al (2009) System-scenario-based design of dynamic embedded systems. ACM Trans Des Autom Electron Syst 14(1):1–45

Goncalves VM, Maia CA, Hardouin L (2015) On the steady-state control of timed event graphs with firing date constraints. IEEE Trans Autom Control 61(8):2187–2202

Goncalves VM, Maia CA, Hardouin L (2017) On max-plus linear dynamical system theory: The regulation problem. Automatica 75:202–209

Hardouin L, Cottenceau B, Shang Y, Raisch J (2018) Control and state estimation for max-plus linear systems. Found Trends Syst Control 6(1):1–116

Heidergott B, Olsder GJ, Van Der Woude J (2014) Max Plus at work: modeling and analysis of synchronized systems: a course on Max-Plus algebra and its applications, 48. Princeton University Press

Komenda J, Lahaye S, Boimond JL, Van Den Boom TJ (2018) Max-plus algebra in the history of discrete event systems. Ann Rev Control 45:240–249

Lahaye S, Boimond JL, Ferrier JL (2008) Just-in-time control of time-varying discrete event dynamic systems in (max,+) algebra. Int J Prod Res 46(19):5337–5348

Liroz-Gistau M, Akbarinia R, Pacitti E, Porto F, Valduriez P (2013) Dynamic workload-based partitioning algorithms for continuously growing databases. Transactions on Large-Scale Data-and Knowledge-Centered Systems XII, Springer, pp 105–128

Menguy E, Boimond JL, Hardouin L, Ferrier JL (2000) Just-in-time control of timed event graphs: update of reference input, presence of uncontrollable input. IEEE Trans Autom Control 45(9):2155–2159

Moré JJ (1978) The Levenberg-Marquardt Algorithm: Implementation and Theory, Numerical Analysis. Lecture Notes in Mathematics 630, Springer, pp 105–116

Nasri I, Habchi G, Boukezzoula R (2014) Use of $(max, +)$ algebra for scheduling and optimization of HVLV systems subject to preventive maintenance. Simul Model Pract Theory 46:149–163

Oda N, Ito M, Shibata M (2009) Vision-based motion control for robotic systems. IEEJ Trans Electr Electron Eng 4(2):176–183

Prou J, Wagneur E (1999) Controllability in the max-algebra. Kybernetika 35:13–24

Sau C et al (2021) Design and management of image processing pipelines within CPS: Acquired experience towards the end of the FitOptiVis ECSEL project. Microprocessors and Microsystems: Embedded Hardware Design (MICPRO), 87, Article 104350

Schafaschek G, Hardouin L, Raisch J (2020) Optimal control of timed event graphs with resource sharing and output-reference update. at-Automatisierungstechnik 68(7):512–528

Sigler L (2003) Fibonacci's Liber Abaci: a translation into modern English of Leonardo Pisano's book of calculation. Springer

Silva GGD, Maia CA (2016) On just-in-time control of timed event graphs with input constraints: a semimodule approach. Discret Event Dyn Syst 26:351–366

Stanczyk J (2016) Max-Plus Algebra Toolbox for Matlab

Stonebraker M, Cetintemel U, Zdonik S (2005) The 8 requirements of real-time stream processing. ACM Sigmod Rec 34:42–47

Tsiamis A, Maragos P (2019) Sparsity in max-plus algebra and systems. Discret Event Dyn Syst 29:163–189

van den Boom TJJ, van den Muijsenberg M, De Schutter B (2020) Model predictive scheduling of semi-cyclic discrete-event systems using switching max-plus linear models and dynamic graphs. Discret Event Dyn Syst 1–35

Van Den Boom TJJ, De Schutter B (2002) Model predictive control for perturbed max-plus-linear systems. Syst Control Lett 45(1):21–33

Van Den Boom TJ, De Schutter B (2004) Modelling and control of discrete event systems using switching max-plus-linear systems. IFAC Proc Vol 37(18):117–122

Van Den Boom TJ, De Schutter B (2006) Modelling and control of discrete event systems using switching max-plus-linear systems. Control Eng Pract 14(10):1199–1211

Van Den Boom TJ, De Schutter B (2006) Dynamic railway network management using switching max-plus-linear models. IFAC Proc Vol 39(12):343–348

Van Den Boom TJ, De Schutter B (2012) Modeling and control of switching max-plus-linear systems with random and deterministic switching. Discret Event Dyn Syst 22(3):293–332

Van Horssen, EP (2018) Data-intensive feedback control: Switched systems analysis and design, Ph.D. Dissertation, Eindhoven University of Technology

Zhang Q, Yang LT, Chen Z, Li P (2018) A survey on deep learning for big data. Inf Fus 42:146–157

Roohallah Azarmi received his B.Sc. and M.Sc. degrees from K. N. Toosi University of Technology, Tehran, Iran, in 2012 and 2015, respectively. He is currently a Ph.D. Researcher with the Electronic Systems group, Department of Electrical Engineering, Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands. His research interests include fractional calculus, control of discrete event systems, model predictive control, PID controller tuning, dead-time systems, process control, and multi-objective optimization.

Mohsen Alirezaei received his Ph.D. in Mechanical Engineering, Robotics and Control in 2011 and was a postdoc researcher at Delft University of Technology in 2012. He was a Senior Scientist in the Integrated Vehicle Safety Department of TNO automotive (2012-2019) and part time assistant professor at Delft University of Technology (2015-2019). He is currently working as a Fellow Scientist at Siemens Industry Software and Services in Helmond and is part time assistant professor at Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands. His research interests are verification and validation of automated and cooperative automated driving and advance driver assistance systems.

Dip Goswami is an Associate Professor in the Electronic Systems group of the Department of Electrical Engineering at Eindhoven University of Technology (TU/e). His research focuses on various design aspects of embedded control systems in resource-constrained domains such as automotive and robotics. He has published in several international journals and conferences in the fields of embedded control systems, robotics and cyber-physical systems, resulting in three best paper awards.

Twan Basten received the M.Sc. and Ph.D. degrees in computing science from Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands. He is currently a Professor with the Department of Electrical Engineering, TU/e. His current research interests include the design of embedded and cyber-physical systems, dependable computing, and computational models. He is a senior member of IEEE and a life member of ACM.