Dynamic Constraints for Mixed-Criticality Systems

Dávid Juhász
TU Wien
Vienna, Austria
Imsys AB
Stockholm, Sweden
david.juhasz@tuwien.ac.at

Axel Jantsch
TU Wien
Vienna, Austria
axel.jantsch@tuwien.ac.at

ABSTRACT

We define quality of service requirements for mixed-criticality systems based on min-plus algebra rather than discrete criticality levels. The requirements (1) unify a spectrum of weakly-hard real-time requirements with strongly-hard real-time and soft real-time as extreme cases and (2) support dynamic tuning of task importance. The paper elaborates the relation to mixed-criticality scheduling theory and weakly-hard real-time systems. The supported timing requirements, computational complexity, and scheduling feasibility are discussed.

CCS CONCEPTS

• **Computer systems organization → Real-time system specification**; *Embedded systems*; System on a chip;

KEYWORDS

mixed-criticality systems, quality of service, weakly-hard real-time systems, task allocation, scheduling, multi-core systems

ACM Reference Format:
Dávid Juhász and Axel Jantsch. 2019. Dynamic Constraints for Mixed-Criticality Systems. In *INTERNATIONAL CONFERENCE ON OMNI-LAYER INTELLIGENT SYSTEMS (COINS), May 5–7, 2019, Crete, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3312614.3312625

1 INTRODUCTION

Real-time (RT) scheduling has a long history with well-established results [18]. In *classic* hard real-time (hRT) systems, no task is allowed to miss any of its deadlines. Most RT systems could, however, tolerate some deadlines being missed in a known and predictable way. The weakly-hard real-time (whRT) concept models RT systems that tolerate a clearly specified degree of missed deadlines [2]. Opposed to whRT systems — allowing deadline misses —, strongly-hard real-time (shRT) systems do not allow tasks to miss any deadline. Soft real-time (sRT) systems allow deadline misses without guarantees about them.

The conservative shRT approach for ensuring reliability of critical RT applications accepts underutilized and over-sized systems for performance guarantees. However, economic considerations favor resource sharing to increase hardware utilization and decrease costs. Research on mixed-criticality systems (MCSs) — a term coined by Vestal [20] in 2007 — has accumulated a large body of results [3] as a way of providing performance guarantees for safe resource sharing among tasks of different importance.

Practical applicability of academic results on MCSs is being discussed and concerns have been raised [1, 6, 7]. WhRT concepts have been applied to provide degraded quality of service (QoS) for low-criticality tasks in overloaded MCSs [8]. We propose a new model of MCSs based on whRT concepts — rather than applying those concepts as an addition to an existing model.

Main contributions: We discuss a formulation of QoS requirements over the number of deadline misses. The proposed formulation has the following novel characteristics:

- requirements are based on the min-plus algebra, which allows continuous control over dynamic parameters;
- a spectrum of whRT requirements is unified with shRT and sRT as extreme cases.

The following properties are discussed:

- computational complexity is analyzed and a window-based approximation is suggested to make implementation feasible;
- considerations for scheduling feasibility are outlined.

The paper is concerned with formulating effective requirements for MCSs and solving the resulting optimization problem is left for future work.

2 MOTIVATION

Our work is motivated by various aspects: robustness of MCSs (Section 2.1), providing whRT guarantees (Section 2.2), and supporting dynamic QoS requirements (Section 2.3).

2.1 Mixed-Criticality Systems

MCSs are associated with mixed-criticality scheduling theory (MCSched). Note, however, the more general topic of MCSs [1, 3]. While we consider MCSs in the general sense, our motivation is based on MCSched and its limitations as follows.

MCSched was established by Vestal's seminal paper [20]. The considered task model is based on sporadic task systems as summarized in [3, Section 2]. A task system consists of tasks. Each task is defined by its period (minimal inter-arrival time), deadline, worst-case execution time (WCET), and criticality level (CL). Some, in rare cases indeed all, of the parameters have CL-dependent values (e.g., separate WCET estimates for CLs). Tasks generate a potentially unbounded sequence of jobs to execute.

The MCS — as in MCSched — starts running at its lowest CL mode and executes jobs for all tasks. At any CL mode, the system executes jobs for tasks with CL not below the current mode. When any executed job violates its CL-dependent WCET in the current mode, CL mode is raised to the next level; lower-CL jobs get ignored.

While there are different variants of the basic model depending on the properties of the parameters and the number of allowed CLs, all variants share common traits: MCSched is focused on *a priori* verification and its support for runtime robustness — the second aspect of safety-critical RT systems — is open to debate [1].

Abandoning tasks with lower CL and never returning to a low CL state is a major issue raised by systems engineers [1, 6, 7]. Some approaches mitigate the issue by reconfiguring the system to abandon tasks in a more graceful manner [3, Section 6].

Compliance to safety standards is important for industrial applications. The delicate connection between CLs and safety assurance levels (safety integrity level (SIL) in IEC 61508, automotive safety integrity level (ASIL) in ISO 26262, or development assurance level (DAL) in DO 178C) is discussed in [6] with arguments against the practical applicability of MCSched. The common ground of concerns is safety standards requiring separation among different assurance levels, which MCSched does not provide among CLs [3, 6, 7].

We also considered suggestions from [9] about safety assurance for MCSs, most importantly: need for partitioning integrity — that is temporal separation in our context — and problem formulation in terms of task importance — that describes the consequence of missing the deadline and varies dynamically.

This Work: We use a sporadic task model without CL-dependent values (i.e., values that are equal for all CLs): a MCS is defined based on task importance as individual QoS requirements, rather than statically-defined discrete CLs. We formulate task allocation with QoS requirements as *constraints* and realize partitioning integrity with respect to a minimal level of required service.

2.2 Weakly-Hard Real-Time Systems

ShRT systems require all deadlines to be met; sRT systems allow deadlines to be missed. sRT systems optimize some metric based on deadline misses but do not provide guarantees on bounding the number and distribution of deadline misses. WhRT systems bridge the gap between shRT and sRT systems by bounding the distribution of met and missed deadlines [2].

MCSched provides *a priori* verification of separate shRT models for each CL. While each CL is considered as a shRT system, switching between CLs renders MCSched a sRT system: allowing low-CL tasks to miss deadlines without bounds. While deadline misses are present in MCSs inherently, their distribution needs to be bounded for providing partitioning integrity.

This Work: We define MCSs as whRT systems by specifying QoS requirements as bounds on deadline misses. Fine-grained setting of task importance is supported by QoS parameters. The parameters unify a spectrum of whRT requirements including shRT and sRT as extreme cases.

2.3 Dynamic Quality-of-Service Requirements

Task importance in MCSs varies dynamically [9]. As far as we know, no MCS model with support for dynamic adjustment of task importance has been proposed yet.

This Work: We define dynamic QoS parameters to enable runtime adjustment of task importance. The QoS requirements are formulated based on the min-plus algebra [13], which allows continuous control of requirements by changing QoS parameters.

3 PRELIMINARIES

The section summarizes the model and notations for defining quality of service (QoS) requirements in Section 4: platform (Section 3.1) and task (Section 3.2) models, properties of jobs and their execution (Section 3.3) and QoS parameters (Section 3.4) are defined.

3.1 Platform Model

We model the platform as a set of processing elements (PEs), Π, with the effects of shared memory and the NoC being implicit. This abstraction limits accuracy without losing generality since QoS *requirements* are independent of those details. The same approach can be taken with full network on chip (NoC) and memory models to improve QoS *guarantees*.

We assume a discrete time model with time granularity $\delta = 1$, which can be considered equivalent to one clock cycle of PEs.

3.2 Task Model

We follow the sporadic task model. A task, τ_i, is defined by its minimal inter-arrival time, relative deadline, and PE-dependent worst-case execution time (WCET) as $(T_i^\tau, D_i^\tau, \vec{C}_i^\tau)$. The WCET $C_i^\tau[q]$ stands for executing task τ_i on PE π_q.

In the case of functional asymmetric multi-cores, $C_i^\tau[q] = \infty$ indicates that τ_i cannot be executed on π_q.

Standard timing conditions (Eq. (1)) hold.

$$\forall \tau_i : \max_{q, C_i^\tau[q] \neq \infty} C_i^\tau[q] \leq D_i^\tau \leq T_i^\tau \tag{1}$$

3.3 Jobs and their Execution

A task τ_i generates a potentially infinite sequence of jobs, $\iota_{i,j}$, with properties inherited from τ_i.

A new job is generated at time $A_{i,j}^\iota$, which is called the admission time of $\iota_{i,j}$. The absolute deadline of the job is then $D_{i,j}^\iota = A_{i,j}^\iota + D_i^\tau$.

The skip flag $S_{i,j}^\iota \in \mathbb{B}$ ($\mathbb{B} = \{\texttt{False}, \texttt{True}\}$) indicates if the job has been skipped, that is never released for execution.

An executed job $\iota_{i,j}$, for which $S_{i,j}^\iota = \texttt{False}$, is mapped on a PE $P_{i,j}^\iota \in \Pi$ and scheduled to be released for execution at $R_{i,j}^\iota \in \mathbb{N}^+$. Once a job starts to execute, it runs for completion; preemption is not supported. Note a natural restriction of task allocation: no PE may execute more than one job at any time.

The completion time $C_{i,j}^\iota$ of $\iota_{i,j}$ is when the job signals completion. The execution time is $E_{i,j}^\iota = C_{i,j}^\iota - R_{i,j}^\iota$. No job is expected to execute longer than its WCET: $E_{i,j}^\iota \leq C_i^\tau[q]$ where $\pi_q = P_{i,j}^\iota$.

The earliest possible admission (EPA) of job $\iota_{i,j+1}$ is $\hat{A}_{i,j+1}^\iota = A_{i,j}^\iota + T_i^\tau$. Job admission respects EPA times, that is $A_{i,j}^\iota \geq \hat{A}_{i,j}^\iota$.

3.4 Parameters for Requirements

Requirements are derived from the following dynamic QoS parameters for each task τ_i:

- $\mathrm{S}_{\tau_i;M}(t) \in \left[\underline{M}_{\tau_i}, \overline{M}_{\tau_i}\right]$ $(0 \leq \underline{M}_{\tau_i} \leq \overline{M}_{\tau_i} \leq 1)$ is the *allowed increase rate* for non-bursting deadline misses to occur with;

- $S_{\tau_i;b}(t) \in \{0, \ldots, \overline{B}_{\tau_i}\}$ is *allowed burstiness*, the limit of consecutive deadline misses occurring in a burst.

The bounds $\underline{M}_{\tau_i}$, $\overline{M}_{\tau_i}$, and $\overline{B}_{\tau_i}$ are static parameters of each task.

The QoS parameters may change their value within their corresponding static bounds at any time. The actual dynamics of the parameters is dependent on application-specific factors beyond the scope of the model. Hence, the changes in the parameters are best perceived as stochastic disturbances.

4 QUALITY OF SERVICE REQUIREMENTS

The section specifies the actual requirements: basic definitions (Section 4.1), the QoS requirements (Section 4.2), and some necessary additional requirements (Section 4.3) are described.

4.1 Definitions for Quality of Service

4.1.1 The Number of Deadline Misses. We use the indicator function $\chi : \mathbb{B} \rightarrow \{0, 1\}$ (Eq. (2)).

$$\chi(\text{expr}) = \begin{cases} 0 & \text{if expr} = \texttt{False} \\ 1 & \text{if expr} = \texttt{True} \end{cases} \tag{2}$$

The number of jobs generated by task τ_i with absolute deadline not later than t is N_{τ_i} (Eq. (3)).

$$N_{\tau_i}(t) = \max\{ j \in \mathbb{N}^+ \mid D^{\iota}_{i,j} \le t \} \tag{3}$$

The delayed cumulative deadline miss function for task τ_i starting at time s is $M_{\tau_i;s}$ (Eq. (4)): the number of jobs that were generated by τ_i and missed their deadlines between times s and $s + t$ (an arbitrary period of the system's lifetime), including skipped jobs.

$$M_{\tau_i;s}(t) = \sum_{j=N_{\tau_i}(s)+1}^{N_{\tau_i}(s+t)} \chi(S^{\iota}_{i,j} \vee C^{\iota}_{i,j} > D^{\iota}_{i,j}) \tag{4}$$

4.1.2 Deadline Miss Bounds. While QoS parameters $S_{\tau_i;M}$ and $S_{\tau_i;b}$ may change at any time, deadline miss bounds are derived on a job-by-job basis. The deadline miss bound for task τ_i is based on the following parameters:

- allowed increase rate of deadline misses for job $\iota_{i,j}$ at $A^{\iota}_{i,j}$ is $Q_{\tau_i;M}(j)$ (Eq. (5));

$$Q_{\tau_i;M}(j) = S_{\tau_i;M}(A^{\iota}_{i,j}) \tag{5}$$

- allowed burstiness of deadline misses for τ_i at time t is $Q_{\tau_i;b}(t)$ (Eq. (6)).

$$Q_{\tau_i;b}(t) = S_{\tau_i;b}(t) \tag{6}$$

The delayed cumulative deadline miss bound for task τ_i starting to count at time s is $\mu_{\tau_i;s}$ (Eq. (7)): the number of allowed deadline misses between times s and $s + t$ for jobs generated by τ_i,

$$\mu_{\tau_i;s}(t) = \left\lfloor \sum_{j=N_{\tau_i}(s)+1}^{N_{\tau_i}(s+t)} Q_{\tau_i;M}(j) \right\rfloor + Q_{\tau_i;b}(s) \tag{7}$$

4.2 Quality-of-Service Requirements

QoS requirements are defined on the number of deadline misses. For respecting the QoS parameters (Section 3.4) both globally and locally, deadline miss requirements are defined by the convolution in min-plus algebra as in network calculus [13].

Min-plus algebra describes linear real-time (RT) systems; detailed description is available in standard textbooks, for example [13]. The min-plus convolution (⊗, Eq. (8)) can be used to calculate the output rate, $a \otimes s$, of a service working with rate s and input arriving at rate a (i.e., rate a is bounded by s as $a \otimes s$).

$$(f \otimes g)(t) = \min_{0 \le s \le t} \{ f(t-s) + g(s) \} \tag{8}$$

We set QoS requirements by bounding deadline misses, $M_{\tau_i;t}$, by the corresponding deadline miss bound function, $\mu_{\tau_i;t}$, (Section 4.1). The actual number of deadline misses of task τ_i is to be constrained by the bounded flow of deadline misses at any time t (Eq. (9)).

$$M_{\tau_i;t} \le \left(M_{\tau_i;t} \otimes \mu_{\tau_i;t}\right) \tag{9}$$

In analogy to network calculus, a flow of deadline misses is bounded by a deadline miss bound function as an arrival curve.

We have the inequalities in Eq. (10) as requirements.

$$\forall \tau_i,\ t : M_{\tau_i;t} - \left(M_{\tau_i;t} \otimes \mu_{\tau_i;t}\right) \le 0 \tag{10}$$

4.3 Notion of Deadline Misses

As QoS is based on deadline misses, we briefly review different notions how deadlines may be missed according to [2]:

delayed completion allows each job to run to completion even though it finishes after the deadline;
abortion terminates any job that missed the deadline;
skip allows the system to *not release* — skip — admitted jobs, the whole invocation is not executed;
rejection allows the system to *not admit* — reject — jobs.

Our model allows jobs to be *skipped* but not to be *aborted* — as preemption is not supported (Section 3.3). *Rejection* is not applicable to our model because QoS requirements are defined over deadline misses of *admitted jobs* (Section 4.1).

Allowing *delayed completion* of released (i.e., not-skipped) tasks let more jobs to be completed at the cost of less room for utilizing dynamic power management (DPM) — features that are considered for future work (Section 5.5). Also, the value contributed by *delayed completion* is application-dependent. Hence, *delayed completion* is not included in our problem formulation.

We require all released jobs to meet their deadlines. Jobs that would not meet their deadlines are to be recognized before release and be *skipped.* This additional requirement is defined in Eq. (11).

$$\forall \tau_i,\ j : \neg S^{\iota}_{i,j} \implies C^{\iota}_{i,j} \le D^{\iota}_{i,j} \tag{11}$$

As a job is either skipped or completed by its deadline and $D^{\tau}_i \le T^{\tau}_i$, not more than one job may be active — neither skipped nor completed — by the EPA of its consecutive job. Since job admission respects EPA times: at most one job may be active in the system for any task at any time.

5 PROPERTIES OF THE MODEL

The section discusses supported real-time (RT) requirements (Section 5.1), continuous control of dynamic requirements (Section 5.2), computational complexity (Section 5.3), scheduling feasibility (Section 5.4), and future work (Section 5.5).

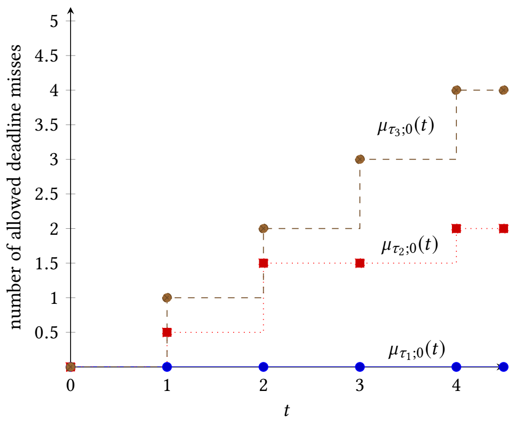

Figure 1: Deadline miss bounds for a shRT task ($\mathrm{S}_{\tau_1;M}(t) = 0$, $\mathrm{S}_{\tau_1;b}(0) = 0$), a whRT task ($\mathrm{S}_{\tau_2;M}(t) = (t/2) \bmod 1.5$, $\mathrm{S}_{\tau_2;b}(0) = 0$), and a sRT task ($\mathrm{S}_{\tau_3;M}(t) = 1$, $\mathrm{S}_{\tau_3;b}(0) = 0$); jobs admitted at each time step starting at $t = 1$

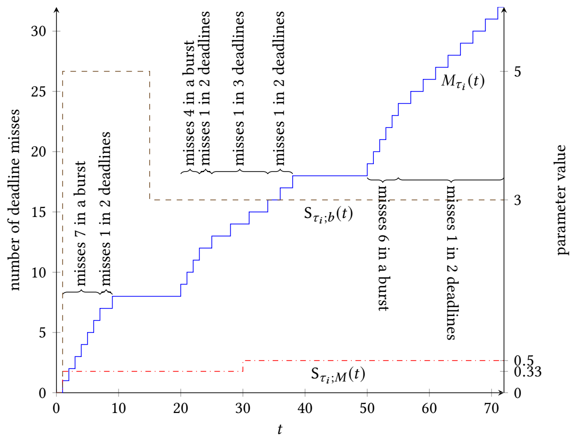

Figure 2: Requirement follows changes of QoS parameters; plot of bounded deadline misses ($M_{\tau_i}(t)$ Eq. (14), left axis) and QoS parameters ($\mathrm{S}_{\tau_i;M}(t)$ and $\mathrm{S}_{\tau_i;b}(t)$, right axis); periods when no deadline is missed let allowed burstiness replenish

Table 1: RT requirements according to possible values of QoS parameters $\mathrm{S}_{\tau_i;M}(t)$ and $\mathrm{S}_{\tau_i;b}(t)$ (invariant in t)

	$\mathrm{S}_{\tau_i;M}(t) = 0$	$0 < \mathrm{S}_{\tau_i;M}(t) < 1$	$\mathrm{S}_{\tau_i;M}(t) = 1$
$\mathrm{S}_{\tau_i;b}(t) = 0$	shRT	whRT (1)	sRT
$\mathrm{S}_{\tau_i;b}(t) > 0$	whRT (2)	whRT (3)	sRT

5.1 Static Requirements

For catching supported RT requirements, consider static quality of service (QoS) parameters: $\mathrm{S}_{\tau_i;M}(t)$ and $\mathrm{S}_{\tau_i;b}(t)$ are invariant in t.

Examples of Different Real-Time Requirements.
Different values of QoS parameters $\mathrm{S}_{\tau_i;M}(t)$ and $\mathrm{S}_{\tau_i;b}(t)$ allow covering a spectrum of weakly-hard real-time (whRT) requirements (see Fig. 1):

strongly-hard real-time (shRT) tasks with $\underline{M}_{\tau_{shRT}} = \overline{M}_{\tau_{shRT}} = 0$ and $\overline{B}_{\tau_{shRT}} = 0$;
whRT tasks with $\underline{M}_{\tau_{whRT}} < 1$ and $\overline{M}_{\tau_{whRT}} > 0$;
soft real-time (sRT) tasks with $\underline{M}_{\tau_{sRT}} = \overline{M}_{\tau_{sRT}} = 1$ and $\overline{B}_{\tau_{sRT}} \geq 0$.

No deadline misses are allowed for shRT tasks and sRT tasks may miss all their deadlines. An acceptable but bounded rate of deadline misses can be specified for whRT tasks between the extremes. Note that we call tasks between the two extremes (i.e., shRT and sRT) whRT tasks, though all requirements — including the extremes — are expressed as whRT requirements.

Inventory of the Supported Real-Time Requirements.
The supported RT requirements are summarized in Table 1. Beyond the extreme cases of shRT and sRT, three types of whRT requirements are supported:

whRT (1) covers a spectrum between shRT and sRT allowing deadlines to be missed regularly: it is $\overline{\binom{1}{m}}$ as "misses any 1 in m deadlines" [2] where $m = \lfloor 1/\mathrm{S}_{\tau_i;M}(t) \rfloor$;
whRT (2) allows $\mathrm{S}_{\tau_i;b}(t)$ deadlines to be missed in total (the allowed burstiness never replenishes): it is $\overline{\langle\, \mathrm{S}_{\tau_i;b}(t) + 1 \,\rangle}$ as "misses row $\mathrm{S}_{\tau_i;b}(t)$+1 deadlines" [2], also $\binom{\mathrm{S}_{\tau_i;b}(t)}{\infty}$ stretching the definition of $\binom{n}{m}$ [2];
whRT (3) allows consecutive deadline misses limited by burstiness $\mathrm{S}_{\tau_i;b}(t)$, which depletes with every missed deadline and replenishes with rate $\mathrm{S}_{\tau_i;M}(t)$: it is $\overline{\langle\, d + 1 \,\rangle}$ as "misses row $d + 1$ deadlines" [2] and $\binom{g}{d+g}$ as "meets any g in $d + g$ deadlines" [2] where d (Eq. (12)) is the number of deadlines that may be missed consecutively until allowed burstiness depletes completely and g (Eq. (13)) is the number of deadlines that must be met consecutively for a depleted burstiness to allow one deadline to be missed.

$$d = \left\lfloor \frac{\mathrm{S}_{\tau_i;b}(t)}{1 - \mathrm{S}_{\tau_i;M}(t)} \right\rfloor \tag{12}$$

$$g = \left\lfloor \frac{1}{\mathrm{S}_{\tau_i;M}(t)} \right\rfloor - 1 \tag{13}$$

5.2 Dynamic Requirements

For discussing dynamic QoS requirements, consider dynamic QoS parameters: $\mathrm{S}_{\tau_i;M}(t)$ and $\mathrm{S}_{\tau_i;b}(t)$ vary in t.

Tracking Dynamically Changing QoS Requirements.
The actual number of deadline misses meeting requirements in Eq. (10) for task τ_i is M_{τ_i} (Eq. (14)). The formula follows from properties of min-plus convolution (Eq. (8), [13]). Requirements in Eq. (10) makes M_{τ_i} follow changing QoS parameters (see Fig. 2) by enforcing deadline miss bounds (Eq. (7)) in *all time windows.*

$$M_{\tau_i}(t) = \min_{0 \leq s \leq t} \left(\left(M_{\tau_i;s} \otimes \mu_{\tau_i;s} \right)(t - s) \right) \tag{14}$$

Task Importance as QoS Parameters.
Task importance as in [9] is an abstract metric of requirement specification; the metric is to be mapped to absolute whRT temporal requirements — QoS parameters — for implementation. Our model supports a spectrum of whRT requirements and allows dynamic changes — in line with [9].

5.3 Computational Complexity

Computing a convolution of t (e.g., $\left(M_{\tau_i;s} \otimes \mu_{\tau_i;s}\right)(t)$) has a time complexity of $\mathcal{O}(t)$. Computing QoS bounds (Eq. (9)) for one task up to t steps has a complexity of $\mathcal{O}(\sum_{w=0}^{t} w)$. Further, computing QoS bounds up to t steps for a task system of N tasks has a complexity of $\mathcal{O}(N \sum_{w=0}^{t} w)$.

The sum $\sum_{w=0}^{\infty} w$ diverges for the infinite requirement in Eq. (10). We recognize that performing infinite computation would render any system infeasible. Note, however, that a corresponding partial sum of W steps has a closed form (Eq. (15)). A window-based approximation of QoS requirements has an overall complexity of $\mathcal{O}(NW^2)$ with a window of length W.

$$\sum_{w=0}^{W} w = \frac{1}{2} W\,(W+1) \qquad (15)$$

While approximating the QoS requirements is imperative for any implementation, we omit detailed discussion of window-based approximation due to space limitation. Conceptually speaking: full accuracy of approximation is being approached as window length — a design parameter — approaches infinity. That is because of a sliding window enforces local requirements; the longer the window, the longer a local requirement is respected.

5.4 Scheduling Feasibility

A system — platform and task set — is *feasible* concerning some requirements if a *feasible* task allocation — that satisfies the requirements — exists. A system is *schedulable* with an algorithm if the algorithm generates a feasible task allocation. Schedulability is a *sufficient* — but *not necessary* — condition of feasibility.

Feasibility.
Considering the strictest ("worst-case") requirements is sufficient to check feasibility of whRT systems with dynamic requirements.

Each task τ_i is considered with QoS parameters $\mathrm{S}_{\tau_i;M}(t) = \underline{M}_{\tau_i}$ and $\mathrm{S}_{\tau_i;b}(t) = 0$ — resulting in strictest requirements:

- shRT tasks with $\underline{M}_{\tau_i} = 0$ must meet all their deadlines;
- whRT tasks with $0 < \underline{M}_{\tau_i} < 1$ may miss 1 in $\left\lfloor 1/\underline{M}_{\tau_i} \right\rfloor$ deadlines (Section 5.1);
- sRT tasks with $\underline{M}_{\tau_i} = 1$ are ignored.

These requirements are $\overline{\begin{pmatrix} 1 \\ \left\lfloor 1/\underline{M}_{\tau_i} \right\rfloor \end{pmatrix}}$ (whRT) and $\overline{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}$ (shRT) of [2]. They correspond to particular *skip factors* [12] and (m, k)-firm deadlines [10] according to [2, Section 3.3].

A necessary condition for feasibility of *occasionally skippable* task sets on *uniprocessors* has been given in [12, Eq. (2)]; while determining feasibility has been proven NP-hard [12, Theorem 3.2].

Schedulability checks for whRT task sets against a few scheduling algorithms have been discussed [2, 11, 12]. The published analyses consider scheduling algorithms that target *uniprocessors* with *static requirements* — properties that do not match our model.

Uniprocessor checks can be adapted for telling *multiprocessor* feasibility (sufficiently but not necessarily). A *uniprocessor* check is applied for each processing element (PE) for every possible mapping. The original complexity is increased by the exponential factor $P^{(I+1)}$ — depending on the numbers of PEs (P) and jobs (I).

Schedulability.
The published scheduling algorithms for whRT systems ([2, 11, 12]) lack support for *multiprocessors* and *dynamic requirements*. No algorithm that is applicable for the task allocation problem at hand is known to us — also see Section 6. Designing and implementing a suitable task allocation algorithm is left for future work.

5.5 Future Work

Our target is an energy-aware dynamic resource management technique for multiprocessor system on chip (MPSoC)-based mixed-criticality systems (MCSs). The discussed QoS requirements are merely a first step of a long journey with the following milestones:

- complete analysis of the proposed QoS requirements on approximation (Section 5.3) and schedulability (Section 5.4);
- a combined problem of task allocation and power management with
  - QoS requirements as *constraints* for partitioning integrity,
  - an energy-aware *objective* that simultaneously optimizes throughput and energy consumption;
- an efficient resource manager for the problem with a window-based approximation of QoS requirements and utilizing dynamic power management (DPM);
- supporting communicating task pairs and communication over a network on chip.

Note that combining whRT requirements and dynamic power policies enables tuning between the contradictory goals of optimizing throughput and energy consumption. Dynamic parameters may be controlled by the appliances themselves based on the situation and system state — see self-adaptive systems [17].

6 RELATED WORK

Mixed-Criticality Scheduling Theory.
Our work is positioned with respect to mixed-criticality scheduling theory (MCSched) in Section 2. Works that provide runtime robustness for MCSched — see examples below — are limited by the basic model (Section 2.1).

Robustness of MCSs is discussed in the context of MCSched [4]. The *severity of timing faults* on *the group of high-criticality tasks* is analyzed. Tolerance against high-criticality task overruns and occasional skipping of *robust* tasks are utilized for graceful degradation.

Control theory has been applied [16] to implement runtime resilience for MCSched. The dual-criticality system is resilient against overruns of high-criticality tasks. Low-criticality tasks are not abandoned but may miss deadlines. Resource bounds can be calculated: QoS guarantees are synthetic properties of the integrated system.

WhRT requirements have been applied to MCSched [8] to reduce the load on the system. Some low-criticality tasks are skipped according to (m, k)-*firm deadlines* [10] when the system is in high-criticality mode — a degraded service instead of abandoning tasks. However, QoS (full or degraded) is unpredictable as being subject to occasional criticality mode switches.

Quality of Service.
The term QoS is used in many different areas. We consider QoS in relation to RT systems. The selected papers are also related to energy-awareness — a relevant aspect for future work.

Most of the papers consider QoS in combination with sRT systems, for example [5, 14]. As sRT systems do not constrain deadline misses, QoS is used as a measure of "profit" that is realized by meeting deadlines. The overall "profit" is to be optimized. QoS for sRT systems does not define any absolute guarantees and hence no partitioning integrity is supported. Dynamic tuning of QoS requirements is typically not supported either.

On the other end of the RT spectrum, shRT systems do not allow flexibility in meeting deadlines. QoS is, nevertheless, used in some cases. For example, QoS classes are used to define the deadline of jobs generated by aperiodic tasks with varying workload [19].

A whRT system with QoS guarantee is described in [15]. QoS is defined as requirement of (m, k)-*firm deadlines* [10]. The proposed model is similar to ours: allowing fine-grained setting of whRT requirements and providing partitioning integrity accordingly. However, dynamic tuning of QoS requirements is not supported.

Weakly-Hard Real-Time Systems.
The whRT scheme of [2] generalizes other static whRT requirements [10, 12, 21]. The connection of the proposed QoS requirements to [2] is discussed in Sections 5.1 and 5.4.

7 CONCLUSION

We define QoS requirements for MCSs as dynamic whRT requirements based on the min-plus algebra. The QoS parameters unify a spectrum of whRT requirements with shRT and sRT as extreme cases. The fine-grained dynamic QoS requirements enable mixed-criticality (MC) problem formulation in terms of task importance. Continuous control over changing requirements is also supported.

The paper motivates and positions the proposed QoS formulation with respect to MCSched and whRT systems. Different properties of the requirements are discussed as well.

We recognize that an approximation of the described ideal QoS requirements is a necessity for implementation. Detailed discussion of a window-based approximation and a task allocation algorithm with schedulability analysis are left for future work.

Our final target is an energy-aware dynamic resource management technique for MPSoC-based MCSs. We consider the presented QoS requirements a basis for a relevant problem definition. We believe that providing proper partitioning integrity and support for dynamic tuning of requirements for both task importance and power policies — features we design our solution around — are going to be imperative for future smart embedded systems.

ACKNOWLEDGMENTS

This research was partially funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 674875 (oCPS Marie Curie Network).

REFERENCES

[1] Sanjoy Baruah. 2018. Mixed-Criticality Scheduling Theory: Scope, Promise, and Limitations. *IEEE Des. Test* 35, 2 (2018), 31–37. https://doi.org/10.1109/MDAT.2017.2766571

[2] Guillem Bernat, Alan Burns, and Albert Liamosi. 2001. Weakly Hard Real-Time Systems. *IEEE Trans. Comput.* 50, 4 (2001), 308–321. https://doi.org/10.1109/12.919277

[3] Alan Burns and Robert I. Davis. 2018. Mixed Criticality Systems — A Review. (2018), 72 pages. https://www-users.cs.york.ac.uk/burns/review.pdf

[4] Alan Burns, Robert I. Davis, Sanjoy Baruah, and Iain Bate. 2018. Robust Mixed-Criticality Systems. *IEEE Trans. Comput.* 67, 10 (2018), 1478–1491. https://doi.org/10.1109/TC.2018.2831227

[5] Eduardo Camponogara, George Lima, Daniel Mossé, and Ríad Nassiffe. 2013. Optimizing QoS in Adaptive Real-Time Systems with Energy Constraint Varying CPU Frequency. In *2013 III Brazilian Symp. Comput. Syst. Eng.* 101–106. https://doi.org/10.1109/SBESC.2013.9

[6] Rolf Ernst and Marco Di Natale. 2016. Mixed Criticality Systems—A History of Misconceptions? *IEEE Des. Test* 33, 5 (2016), 65–74. https://doi.org/10.1109/MDAT.2016.2594790

[7] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. 2015. How realistic is the mixed-criticality real-time system model?. In *Proc. 23rd Int. Conf. Real Time Networks Syst. - RTNS '15*. ACM Press, 139–148. https://doi.org/10.1145/2834848.2834869

[8] Oliver Gettings, Sophie Quinton, and Robert I Davis. 2015. Mixed criticality systems with weakly-hard constraints. In *Proc. 23rd Int. Conf. Real Time Networks Syst. - RTNS '15*. ACM Press, 237–246. https://doi.org/10.1145/2834848.2834850

[9] Patrick Graydon and Iain Bate. 2013. Safety Assurance Driven Problem Formulation for Mixed-Criticality Scheduling. In *Proc. Work. Mix. Syst.* 19–24.

[10] Moncef Hamdaoui and Parameswaran Ramanathan. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Trans. Comput.* 44, 12 (1995), 1443–1451. https://doi.org/10.1109/12.477249

[11] Jian Li, YeQiong Song, and Françoise Simonot-Lion. 2004. Schedulability analysis for systems under (m,k)-firm constraints. In *IEEE Int. Work. Fact. Commun. Syst. 2004. Proceedings.* IEEE, 23–30. https://doi.org/10.1109/WFCS.2004.1377670

[12] G. Koren and D. Shasha. 1995. Skip-Over: algorithms and complexity for overloaded systems that allow skips. *Proc. 16th IEEE Real-Time Syst. Symp.* (1995), 110–117. https://doi.org/10.1109/REAL.1995.495201

[13] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network Calculus.* Lecture Notes in Computer Science, Vol. 2050. Springer Berlin Heidelberg. xix – 274 pages. https://doi.org/10.1007/3-540-45318-0

[14] Jinkyu Lee, Insik Shin, and Arvind Easwaran. 2010. Online robust optimization framework for QoS guarantees in distributed soft real-time systems. In *Proc. tenth ACM Int. Conf. Embed. Softw. - EMSOFT '10.* ACM, 89–98. https://doi.org/10.1145/1879021.1879034

[15] Linwei Niu. 2010. Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee. In *2010 IEEE 16th Int. Conf. Embed. Real-Time Comput. Syst. Appl.* 163–172. https://doi.org/10.1109/RTCSA.2010.41

[16] Alessandro Vittorio Papadopoulos, Enrico Bini, Sanjoy Baruah, and Alan Burns. 2018. AdaptMC: A Control-Theoretic Approach for Achieving Resilience in Mixed-Criticality Systems. In *30th Euromicro Conf. Real-Time Syst. (ECRTS 2018)*, Vol. 106. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 14:1–14:22. https://doi.org/10.4230/LIPIcs.ECRTS.2018.14

[17] Jurgo S. Preden, Kalle Tammemäe, Axel Jantsch, Mairo Leier, Andri Riid, and Emine Calis. 2015. The Benefits of Self-Awareness and Attention in Fog and Mist Computing. *Computer (Long. Beach. Calif).* 48, 7 (2015), 37–45. https://doi.org/10.1109/MC.2015.207

[18] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K Mok. 2004. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Syst.* 28, 2-3 (2004), 101–155. https://doi.org/10.1023/B:TIME.0000045315.61234.1e

[19] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron, and Zhijian Lu. 2003. Power-aware QoS management in Web servers. In *RTSS 2003. 24th IEEE Real-Time Syst. Symp. 2003.* IEEE Comput. Soc, 63–72. https://doi.org/10.1109/REAL.2003.1253254

[20] Steve Vestal. 2007. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *28th IEEE Int. Real-Time Syst. Symp. (RTSS 2007).* IEEE, 239–243. https://doi.org/10.1109/RTSS.2007.47

[21] Horst F. Wedde and Jon A. Lind. 1997. Building Large, Complex, Distributed Safety-Critical Operating Systems. *Real-Time Syst.* 13, 3 (1997), 277—-302. https://doi.org/10.1023/A:1007915628098