Mixed-Criticality Control System with Performance and Robustness Guarantees

Long Cheng†, Kai Huang§, Gang Chen¶, Biao Hu† and Alois Knoll†
†Technical University Munich, Germany §Sun Yat-Sen University
¶Northeastern University of China
Email: †{chengl, hub, knoll}@in.tum.de §huangk36@mail.sysu.edu.cn
¶chengang@cse.neu.edu.cn

ABSTRACT

Nowadays, many embedded systems consist of a mix of control applications and soft real-time tasks. This paper studies how to ensure the worst-case quality of control for control applications under disturbances while providing maximal resource to soft real-time tasks. To solve this problem, we propose a mixed-criticality control system model in which the tasks can switch between two operating modes, **LO** and **HI**, according to controlled plant states. In **HI** mode, the worst-case qualities of control to plants are guaranteed, while in **LO** mode, system resources are balanced between two classes of tasks. We compare our approach with other two approaches in the literature. Case study results demonstrate the effectiveness of our system model.

1. INTRODUCTION

Recently, more and more embedded systems comprise a mix of control applications with requirements for control performance as well as robustness, and soft real-time applications. The control applications are usually crucial for the safety of embedded systems. Failing to meet their performance or robustness requirements could cause catastrophes, even human lives. The soft real-time applications are non-crucial for the system safety, and temporal violations of their deadlines only degrade the QoS (Quality of Service) of the system, so-called soft real-time applications. Both classes of tasks together form a mixed-criticality system (MCS) where applications with different levels of criticality are implemented and compete for available resources in the system.

The main concern of the MCS is how to guarantee the quality of control (QoC) of control applications, where QoC is a short term for the control cost and apparent phase margin introduced in section 2. This problem is, however, non-trivial. The reason is, in reality, it is difficult to accurately model and predict the actual environment of the system. The controlled plants may occasionally work in unsafe states due to unexpected disturbances from the environment. A design taking the worst-case scenario into consideration can prevent the plants from failure. However, this can lead to an inefficient utilization of system resources because significant amount of resource is reserved for the worst-case scenario, which rarely happens. Therefore, it is a challenging problem to design an efficient system model and the scheduling scheme so that they not only guarantee the required worst-case QoC (WC-QoC) to plants but also provide sufficient service to soft real-time tasks in normal cases.

In this paper, we propose a mixed-criticality control system (MCCS) model that has two operation modes, **LO** (low) and **HI** (high), to solve this problem. Although MCS is not new in the literature, directly using classical MC models and methods [5] in this context is not applicable due to following reasons:

- Classical MCSs are deployed to strictly guarantee the timing correctness of high-criticality tasks. Therefore, the system changes to **HI** model once the execution time of any high critical task exceeds the worst-case execution time (WCET) in **LO** mode [5]. However, only considering the time correctness is not enough to guarantee the QoC of a control application, as it is influenced by many factors, including the sampling period, control-law, sampling-actuator delay, etc [2, 3, 7].
- In classical MCS models, the periods of high-criticality tasks are the same in both modes. As mentioned above, the QoC of a control application is influenced by the periods. Therefore, the sampling periods of control applications may need to be adjusted in order to provide the required WC-QoC.

In summary, to ensure the WC-QoC of control applications, our mixed-criticality control system model needs to consider a set of new features, compared with classical MCS: (1) Our system switches between two operation modes depending on plant states, i.e., if any plant is in a critical state. (2) To provide the necessary QoC, the period configurations in LO and HI modes could be different. (3) In **HI** mode, the control-laws are also changed so that it is optimal for the period configuration in **HI** mode. 4) Moreover, the corresponding WCETs could also be different, depending on the control-laws.

The MCCS starts with its mode being **LO**. In **LO** mode, the control applications have the optimal QoC under the deadline constraints of soft real-time tasks. Once one or more plant is in its unsafe states, the MCCS switches to **HI** mode. In **HI** mode, control applications are guaranteed with pre-designed performance and robustness for the safety operation. Generally, the periods in **HI** mode need to be decreased for higher QoC. Meanwhile, the periods of soft real-time tasks are amplified by a common factor so that the control applications are still schedulable during mode change as well as in **HI** mode. In this way, the WC-QoC of control applications are ensured in **HI** mode while the soft real-time tasks only suffer a temporary but graceful service degradation. The contributions of this paper can be summarized as:

- We propose a WC-QoC-guaranteed system model for the mixed-criticality systems composing of control and real-time tasks. This model can guarantee the control applications pre-given QoC so that the plants work safely in worst-case scenarios in which unexpected disturbance is applied.
- An approach is proposed to determine the allocation of system resources in **HI** mode so that soft real-time tasks can obtain the maximal resources while constraints of control applications are met.
- The stability of controlled plants are discussed to ensure the stability of plants during mode changes under environment disturbances.

Related Work There is little work on the joint scheduling of a system comprising control and soft real-time tasks. Goswami and et al. proposed an approach that optimizes control performance and respects the timing requirements of the real-time applications in [9]. They assume the system has only one operation mode in the work, which could restrict the quality of control for control applications in worst cases. There has been some work on multi-mode or multi-frequency controllers. In [8], Liu and et al. presented a software fault tolerance architecture for real-time control systems named On-demand Real-TimE GuArd (ORTEGA). Buttazzo and et al. proposed an elastic model in [6] for controllers in which task periods are treated as springs. The stability of systems to switch between multi-mode has also been studied [17]. Wang and et al. proposed an approach offering optimal switching laws for periodic switching systems in [21]. While having made great contribution is this field, the aforementioned work doesn't consider the effect of disturbance and sampling noises to plants. Moreover, the control laws are also not investigated. In this work, the LQG controller is adopted to handle the disturbance and noises. The control laws in **HI** mode are also investigated so that the stability during mode change as well as control performance and robustness are ensured. Plenty of work can be found on the control scheduling synthesis for merely control applications [1, 2, 12, 13, 15, 19]. While achieving considerable progress in improving the control performance, their work doesn't consider the co-scheduling problem of control and soft real-time tasks, which is one of the contributions of this work.

The rest of this paper is organized as follows. Section 2 introduces system models and the operation mode change mechanism. Section 3 demonstrates the motivation. Section 4 introduces the background of our approach and defines the problem. Section 5 discuss the algorithm for system re-configuration. Section 6 and section 7 investigate the stability and the schedulability of the system during mode change, respectively. The proposed approach is evaluated in section 8 and section 9 concludes.

2. SYSTEM MODELS

2.1 Plant Model

A set of plants $\mathbf{P}$ is considered in our system. The number of plants is denoted as m. Each plant P_i is modeled as a continuous-time linear time-invariant (LTI) system which is described by

$$\begin{aligned} \dot{\mathbf{x}}_i &= \mathbf{A}_i\mathbf{x}_i + \mathbf{B}_i\mathbf{u}_i + \mathbf{v}_i \\ \mathbf{y}_i &= \mathbf{C}_i\mathbf{x}_i + \mathbf{e}_i \end{aligned} \tag{1}$$

where $\mathbf{x}_i$, $\mathbf{u}_i$, and $\mathbf{y}_i$ are the plant state, control signal and the continuous-time output, respectively. The control signal is updated periodically and held constant until the next signal. $\mathbf{v}_i$ denotes the process noise and unexpected environmental disturbance. The plant output is sampled periodically with the measurement noise $\mathbf{e}_i$, which is discrete-time Gaussian white-noise. We assume the process and sample noises are both zero-mean white-noise and modeled by given covariance matrices. The matrices $\mathbf{A}_i$ and $\mathbf{B}_i$ indicate how the plant state evolves with the current state and control signal. The matrix $\mathbf{C}_i$ models the physical sensors measuring the plant state. Moreover, a control-law is required to calculate the control signal $\mathbf{u}_i$ from the output of the plant.

Due to the sample and process noises, it is very hard to get the actual state of plants. Therefore, each plant is controlled by a LQG controller, which comprises a Kalman filter [3] offering the estimation of current state, $\hat{\mathbf{x}}_i$, to compute the control signal. Besides, a prediction of next state $\bar{\mathbf{x}}_i$ is also provided by the Kalman filter, which is utilized in following content.

In this paper, two types of workspace are defined for each plant P_i: safe workspace W_i^s and critical workspace W_i^c. The safe workspace of a plant is defined as the workspace in which it behaves in an expected normal manner. However, in the real world, due to unexpected disturbance or system faults, a plant may jump out of its safe workspace to the critical workspace. The critical workspace is defined as the workspace in which the plant behaves unsafely in an unexpected manner or is going to be unstable. Without a loss of generality, a state-checking function, $F_i^N(\hat{\mathbf{x}}_i)$ is defined to indicate whether P_i is in its safe workspace or critical space. The function is assumed to have only two scalar values: 0 and 1, where 0 represents the critical space while 1 represents the saft space. The specific definitions of $F_i^N(\hat{\mathbf{x}}_i)$ are determined by the designer, depending on the specific requirements of the plant.

2.2 Platform and Application Model

We consider a uniprocessor MCCS scheduled by fixed-priority, preemptive scheduling in this paper. The MCCS has two operation modes, **LO** and **HI**, and executes a dual-criticality task set containing m control applications for plants $\mathbf{P}$, and n soft real-time tasks (SRTT). Each plant P_i is controlled by a control application Λ_i which comprises a control law for the plant and a task τ_i running in the system. Then, the whole task set is defined as $\mathbf{T} = \{\tau_1, \tau_2, \cdots, \tau_{m+n}\}$. For clarity, task set $\mathbf{\Theta} = \{\tau_1, \cdots, \tau_m\}$ and $\mathbf{\Upsilon} = \{\tau_{m+1}, \cdots, \tau_{m+n}\}$ denote the tasks corresponding to the m control applications and the n soft real-time tasks, respectively. We consider all control tasks as **HI** criticality and soft real-time tasks as **LO** (low) criticality.

In this paper, we assume the deadline of a task $\tau_i \in \mathbf{T}$ equals the period. Then, the control application Λ_i can be abstracted by a tuple: $\{[H_i(\mathtt{HI}), C_i(\mathtt{HI}), \chi_i(\mathtt{HI})], [H_i(\mathtt{LO}), C_i(\mathtt{LO}), \chi_i(\mathtt{LO})]\}$, where

- $H_i(\lambda) \in \mathbf{R}^+$ is the arrival period of task τ_i in mode λ
- $C_i(\lambda) \in \mathbf{R}^+$ is the WCET of task τ_i in mode λ
- $\chi_i(\lambda)$ is the control law of plant P_i in mode λ

Moreover, during runtime, each task τ_i is assigned an unique priority $p_i(\mathtt{LO})$ (or $p_i(\mathtt{HI})$) if the MCCS is in **LO** (or **HI**) mode. In this paper, the priorities of tasks are ordered in the way that a smaller value indicates a higher priority. For brevity, we define the execution configuration of a control task τ_i:

$$\textit{config}_i(\mathtt{LO}) = \{H_i(\mathtt{LO}), C_i(\mathtt{LO}), \chi_i(\mathtt{LO}), p_i(\mathtt{LO})\} \tag{2}$$

$$\textit{config}_i(\mathtt{HI}) = \{H_i(\mathtt{HI}), C_i(\mathtt{HI}), \chi_i(\mathtt{HI}), p_i(\mathtt{HI})\} \tag{3}$$

Similarly, the execution configuration of a soft real-time task τ_j can be specified by definition (2) and (3) without χ_i.

In **LO** mode, all tasks run at their $\textit{config}(\mathtt{LO})$ configuration, which can be obtained by solving the problem of optimizing the control performance of control applications under the deadline constraints of soft real-time tasks. An existing solution for this problem can be found in [9]. In **HI** mode, the main difference is that we regulate the priorities of control applications higher than those of soft real-time tasks so that control applications provide better QoC. Besides, we also let $H_i(\mathtt{HI}) \leq H_i(\mathtt{LO})$ for the same purpose. It is worth noting that in **HI** mode, soft real-time tasks are guaranteed with degraded service parameters (longer periods and lower priorities) instead of being abandoned. The periods and deadlines of tasks in $\mathbf{\Upsilon}$ are scaled by a scaling factor x ($x \geq 1$).

In conclusion, the following conditions are satisfied:

$$H_i(\mathtt{HI}) \leq H_i(\mathtt{LO}),\ 1 \leq p_i(\mathtt{HI}) \leq m \quad \text{if } \tau_i \in \mathbf{\Theta} \tag{4}$$

$$H_i(\mathtt{HI}) = xH_i(\mathtt{LO}),\ C_i(\mathtt{HI}) = C_i(\mathtt{LO}) \quad \text{if } \tau_i \in \mathbf{\Upsilon} \tag{5}$$

2.3 Operation Mode Change

Our system architecture is illustrated in Fig. 1. After sampling the states of a plant, our system runs the decision module, which

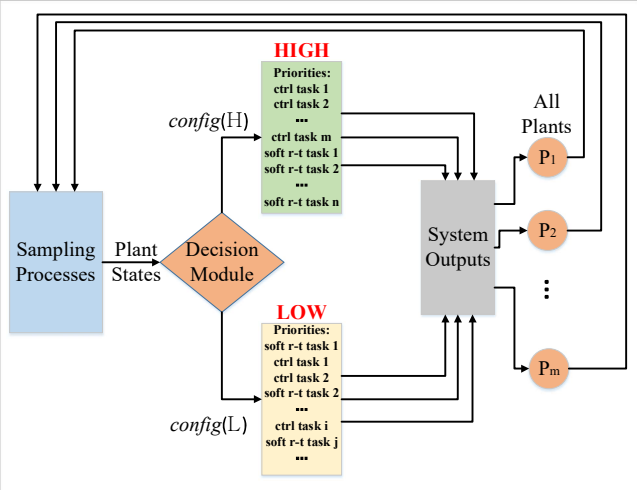

Figure 1: System architecture. Note that for simplicity, notation *ctrl task* indicates control task and notation *soft r-t task* means soft real-time task.

evaluates the states of the sampled plant and decides whether to switch the system to **HI** or **LO** mode. If a mode change decision is made, the system switches to the target mode with the corresponding configuration, which is pre-calculated in design phase. As aforementioned, the configuration in **LO** mode can be obtained using existing approaches. How to calculate the configurations for **HI** mode is discussed in section 5. Then, the system resources are distributed according to the configuration and computes control commands for the plants based on the control-law. Finally, the control commands are sent to the plants.

The mode change decision is made based on the state-checking functions and stability functions F_i^s. The stability function F_i^s of a plant P_i is given to ensure the stability during mode change. It also has two scalar values: 0 and 1. Value 0 means the current mode is **LO** and the future states of the plant will go outside the stability region defined in section 6.2. MCCS needs to change to **HI** mode immediately to handle the case. In **HI** mode, F_i^s gives value 1 if the the plants are stable and stay in their equilibrium points for a certain time length. This time length is an auxiliary parameter defined by the designer to prevent the system from frequently changing the operation mode. When F_i^s have value 1, MCCS can determine whether to switch back to **LO** according to state-checking functions.

In conclusion, MCCS immediately transits from **LO** mode to **HI** mode once the following condition holds:

$$\exists 1 \leq i \leq m, \quad F_i^N(\mathbf{x}_i) F_i^s = 0 \tag{6}$$

MCCS transits back to **LO** mode if all the plants have been in their safe workspaces as well as equilibrium points:

$$\forall 1 \leq i \leq m, \quad F_i^N(\mathbf{x}_i) F_i^s = 1 \tag{7}$$

Once Eq. 6 is satisfied, the system switches to **HI** operation mode. The current **LO** mode control-laws are replaced by **HI** mode control-laws immediately. The already sampled plant states will be dropped. New **HI** control tasks with different priorities are released at the same time [8]. According the the priority, new control tasks samples the state of the plants again and compute control commands according to **HI** mode control-laws. The execution of **LO** soft real-time tasks is paused. Their periods and deadlines are scaled by a scaling factor x to ensure system schedulability during mode change.

Switching from **HI** to **LO** operation mode is a similar procedure except there is a delay in making the real action of mode change. After the decision is made, MCCS keeps operation in **HI** mode for a period of time until all of the control tasks currently being

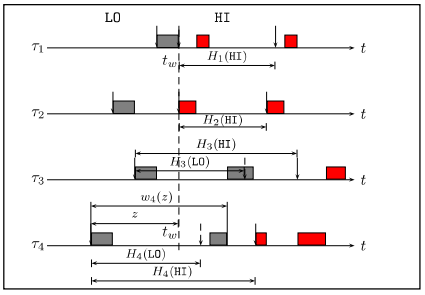

Figure 2: An example of mode change from **LO** to **HI**. Down arrows indicate task arrivals. Specified as the dashed line, t_w is the time instance when the mode change takes place. Gray blocks and red blocks suggest the execution of tasks arriving before and after t_w, respectively.

executed are finished. Since we consider the deadlines of a control task equal the period, the schedulability test ensures no new control tasks arrive in this delay interval. Moreover, the current soft real-time tasks also keep their periods $H(\mathbf{HI})$ until the next arrivals.

Figure 2 shows an example of a mode change from **LO** to **HI** of a system having two control tasks (τ_1 and τ_2) and two soft real-time tasks (τ_3 and τ_4). In **LO** mode, the priorities of τ_1 to τ_4 are assigned as $[1,3,2,4]$. At the transition time t_w, i.e., after sampling the states of P_1 and evaluating Eq. 6, all tasks switch from **LO** to **HI** mode. The control tasks, τ_1 and τ_2, release **HI** mode jobs immediately to replace their **LO** mode jobs. The unfinished jobs of τ_3 and τ_4, however, are not abandoned. Their current deadlines as well as periods are increased by multiplying the same factor 1.5 to save CPU times for control tasks. In **HI** mode, the priorities of the tasks are reassigned to $[2,1,4,3]$. Either τ_1 or τ_2 has a higher priority than both τ_3 and τ_4.

3. MOTIVATION EXAMPLE

Let's consider a system running one control application associated with an inverted pendulum and four soft real-time tasks. The periods and WCETs of the soft real-time tasks are given as $[25,10,30,60]ms$, and $[5,1.5,4.5,12]ms$, respectively. Their deadlines are supposed to be same with periods. We consider the inverted pendulum may suffer impulse disturbances up to $0.6N \cdot sec$ at some unknown time. We also regulate the control signal, i.e., the force, is less than $30N$. Given two control applications Λ_A and Λ_B whose sampling periods are $10ms$ and $2ms$ respectively, our goal is to design the system so that the peak vertical pendulum angle is minimized. The WCET of the control application is given as a constant $1ms$. For demonstration purposes, we exert a $0.6N \cdot sec$ impulse to the pendulum at time $t = 1s$.

Let's first consider the classical approach that our system runs in a single operation mode. Since the utilization of soft real-time tasks is 0.7, application Λ_B cannot be applied, otherwise the total utilization will be 1.2, indicating our system is unschedulable. Therefore, the solution of control application has to be Λ_A. However, as shown in Fig. 3, the pendulum becomes unstable after the impulse is applied, which means this solution is infeasible.

Now, we use our two-operation-mode system model to control the inverted pendulum. Control applications Λ_A and Λ_B are applied in **LO** mode and **HI** mode, respectively. The system switches to **HI** mode to implement Λ_B once the absolute number of the pendulum angle is larger than 5 degrees. The soft real-time tasks run at longer periods in this mode to make the system schedulable. Moreover, the system can switch back to **LO** mode with Λ_A if the angle has been in $[-5,5]$ degrees for one second, returning CPU time back to real-time tasks so that they can provide normal service.

After simulating the system in MATLAB Simulink, we report

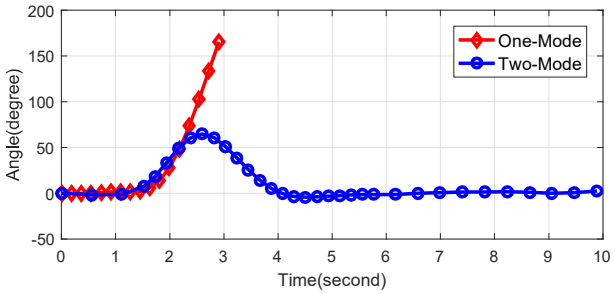

Figure 3: How the angle of the inverted pendulum evolves in two systems: one-operation-mode system and two-operation-mode system.

how the pendulum angle varies in the two cases in Fig. 3. In the simulation, the control application is assigned the highest priority in both modes. From Fig. 3, one can observe that the pendulum is still stable after the impulse in our approach, because the pendulum is controlled by Λ_B after the impulse. Application Λ_B offers much better QoC since its sampling period is smaller. Although the soft real-time tasks suffer service penalty from about $t = 1s$ to $t = 5s$, our approach makes the system work safely with unexpected disturbance, which motivates us to apply the method of mode change to mixed-criticality systems including control and soft real-time tasks.

4. BASIC DEFINITIONS

4.1 Delay and Jitter

In this paper, we assume a control task is activated periodically. The corresponding plant is sampled when the task is released. The output signal is applied to the plant once the task is finished as we assume no communication delay between the controller and plant.

Due to interferences from high-priority tasks and the execution of itself, each control task experiences a delay from sampling to actuation, namely the input-output delay. This delay could be divided into nominal sensor-actuator delay, sensor jitter, and actuator jitter, as an example [2]. In order to simplify the problem, we restrict ourselves to divide the input-output delay into only two parts: a constant delay (L) and a time-varying part, namely the jitter (K) [7]. Fig. 4 graphically illustrates the definition of these two parameters. The delay from sampling and actuation is hence located in the interval $[L, L+K]$. These two parameters of a control task τ_i can be obtained by implementing response-time analysis:

$$L_i = R_i^b \tag{8}$$

$$K_i = R_i^w - R_i^b \tag{9}$$

where R_i^b and R_i^w are the best and worst-case response time of task τ_i. They can be calculated by applying classical response time analysis for fixed-priority [18].

We shall also consider the influence from mode change to the delay and jitter in **HI** mode. Let Δt denote the minimal interval between the sampling of a **LO** mode task and the mode change. This interval is the necessary time needed from sampling data to making a decision on mode change. Then, the best and worst-case response times during mode change are:

$$R_i^b(\mathtt{HI}) = \Delta t + R_i^b \tag{10}$$

$$R_i^w(\mathtt{HI}) = H_i(\mathtt{LO}) + R_i^w \tag{11}$$

The revised response times should be used to calculate the L_i and K_i in **HI** mode so that mode change effect is considered.

4.2 Expected Control Performance

The standard quadratic cost function [3] is used to capture the

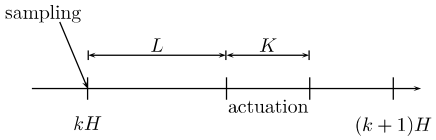

Figure 4: The constant input-output delay and input-output jitter

expected performance of a control application Λ_i:

$$J_i = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}\left\{ \int_0^T \begin{bmatrix} \mathbf{x}_i \\ \mathbf{u}_i \end{bmatrix}^\top Q_i \begin{bmatrix} \mathbf{x}_i \\ \mathbf{u}_i \end{bmatrix} dt \right\}. \tag{12}$$

where $\mathbb{E}\{\bullet\}$ indicates the expected value, and Q_i is a positive semi-definite weight matrix given by the designer. A lower value of J_i indicates a better control quality. For a given plant and its control law, the expected cost J_i can be computed by the toolbox Jitterbug, as long as the input-output (sensor-actuator) delay distribution is known [10].

It is worth noting that the above cost function cannot provide a hard guarantee of stability in the worst case [2]. The next section presents how to obtain robustness or stability results that are valid in the worst case.

4.3 Worst-Case Robustness

Since the expected cost function cannot ensure the worst-case stability of a control application, we introduce the apparent phase margin (APM) to qualify the robustness as well as stability performance of a system. Apparent phase margin is similar to the phase margin in classical control theory. The advantage of (APM) is the influence of the constant delay L_i and the jitter K_i to the stability are taken in account. This makes it suitable for digital control systems. Let notation φ_i^m denote the apparent phase margin of a control application. The stability of plant P_i cannot be guaranteed if $\varphi_i^m \leq 0$. A larger value of φ_i^m suggests a higher degree of robustness. To calculate φ_i^m, we take the plant model P_i, the control application with corresponding sampling period, the best (constant) input-output delay L_i and the worst-case input-output jitter K_i as inputs. Therefore, the apparent phase margin is specified by a function:

$$\varphi_i^m = PM(P_i, \Lambda_i, L_i, K_i). \tag{13}$$

For the formal definition and details of calculating φ_i^m, we refer to [7].

4.4 Control Law Optimizing

For a given plant P_i, a sampling period H_i, and a constant input-output delay, one can find the control-law Φ_i that can provide the best expected quality of control [3]. However, in our system, the execution of lower-priority tasks may be preempted by higher-priority tasks, hence the input-output delay of a task, in reality, is time-varying. In this paper, we first use our real-time simulation toolbox to get the input-output delay distribution of a task. Then, the expected input-output delay can be easily obtained and is used to calculate the control-law by utilizing the toolbox Jitterbug. Finally, we get the Linear-Quadratic-Gaussian (LQG) controller, which is optimal with respect to the expected control performance (Eq. 12), to compensate for the expected input-output delay.

4.5 Problem Definition

In previous sections, the mixed-criticality control system model is introduced. Now, the remaining problem is how to determine the execution configuration in **HI** mode so that: (1) the control costs and APMs of all control applications in **HI** mode meet the requirements given by designer; (2) MCCS is stable during mode

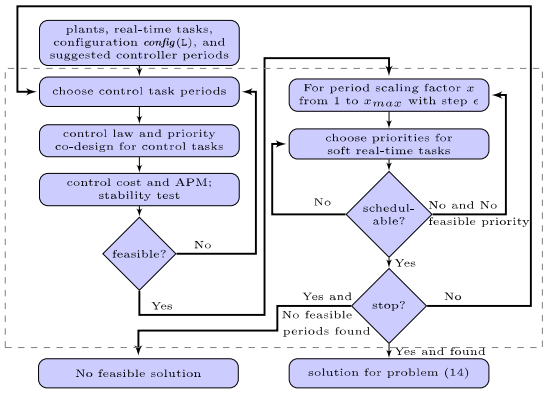

Figure 5: Approach for re-configurating

change; (3) the service degradation of real-time tasks, i.e., the factor x, is minimized.

We term the requirements given by designer for **HI** mode as control cost constraint set $\overline{\mathbf{J}} = \{\overline{J_1}, \cdots, \overline{J_m}\}$ and APM constraint set $\overline{\varphi^{\mathbf{m}}} = \{\overline{\varphi_1^m}, \cdots, \overline{\varphi_m^m}\}$, respectively. Now the problem can be formulated as:

$$\begin{array}{rll} \textbf{minimize} & \text{scaling factor } x & (14) \\ \textbf{subject to} & \forall i \in \{1,\cdots,m\} \quad J_i(\mathtt{HI}) \le \overline{J_i} \text{ and } \varphi_i^m(\mathtt{HI}) \ge \overline{\varphi_i^m} & \\ \textbf{outputs} & \boldsymbol{config}(\mathtt{HI}) = \{\boldsymbol{config}_i(\mathtt{HI}) | i \in \{1,\cdots,m\}\} & \end{array}$$

Note that since we assume the system configuration in **LO** mode is already known, this problem can be solved in early design phase. When the system switches from **LO** mode to **HI** mode in run-time, it directly uses the pre-computed configuration to allocate the resources. Next, an approach and an algorithm are presented to solve this problem.

5. SYSTEM RE-CONFIGURATION

The overall procedure to obtain the execution configuration for **HI** mode is illustrated in Fig. 5. The part surrounded by dashed lines is the loop to optimize controllers' **HI** mode periods (section 5.1). In each iteration, each control application is given a period chosen by the period optimization loop from the given period set $\widehat{\mathbf{H}}_i$. Then, we check whether a priority assignment for control tasks exists so that their control quality and robustness constraints (14) can be guaranteed (section 5.2). The optimal control law for each application is also designed in this step and then the **HI** mode LQG controller is checked by the stability constraint (24) in section 6.2. If a feasible priority assignment can be found, periods and priority assignments for soft real-time tasks are performed to find the period scaling factor x. Finally, the controller period optimization loop determines whether or not the loop should be stopped according to terminating conditions.

5.1 Control Tasks Periods Optimization

The controller periods optimization is accomplished based on the coordinate search method [22] and pattern search method [11]. The optimization is carried out in two steps. Firstly, we utilize the coordinate search to find a promising region for the minimal scaling factor x in the search space. The searching starts from the point where each application has the smallest period in the set. Since a smaller controller period usually leads to a larger x, at each iteration step we allocate a larger period to the controller that currently has the smallest period if its control quality and robustness constraints (14) are met. Note that once x cannot be further minimized by coordinate search, the pattern search is adopted to get the promising point based on the information obtained in the first step. The optimization toolbox in MATLAB is utilized to implement the pattern search.

5.2 Control Tasks Priority Assignment

After a period set is chosen for control tasks, algorithm 1 shows whether one can find a priority assignment for control tasks so that constraint (14) is satisfied. The algorithm is recursive and guided by two parameters, the set of remained plants **RP** and the number of remaining plants N_{rp}. At the first call, we assign them as **P** and m, respectively. Moreover, it also takes the plant models, number of plants m, WCETs, $\overline{\mathbf{J}}$ and $\overline{\varphi^{\mathbf{m}}}$ as inputs. The basic idea is to find a plant that can be assigned the lowest priority while its **HI** mode constraint (14) is still satisfied (line 13). If no plant is found, the algorithm returns an error indicating current control periods are unfeasible (line 11). Otherwise, we recursively call this algorithm with the remaining plants (line 19). Once each control task is assigned a unique priority (line 1), a final check is performed to correct the expected input-output delays calculated in previous steps (line 3), which may be inaccurate in the final priority assignment. With the revised expected input-output delays, new control laws are designed to compensate for the delay (line 5) and the constraint (14) is finally tested (line 6). The algorithm terminates once a feasible priority assignment is found (line 20) or all the possible assignments are checked.

Algorithm 1 Function: ControlPriority(**RP**, N_{rp})

Input: plants **P**, m, controller periods $\widehat{\mathbf{H}}$,WCETs, $\overline{\mathbf{J}}$, $\overline{\varphi^{\mathbf{m}}}$
Output: feasible flag $\mathbb{F}$, $\mathbf{A} = \{\boldsymbol{config}_i(\mathtt{HI}) | i \in \{1,\cdots,m\}\}$

1: **if RP** $== \varnothing$ **then**
2: $\mathbb{F} = True$, $\mathbf{A} = \varnothing$
3: simulation to get expected input-output delays $\mathbf{L}^e$ of all control applications
4: **for** $i = 1$ to m **do**
5: get the optimal control law for P_i, calculate J_{Λ_i} and φ_i^m
6: **if** $J_{\Lambda_i} > \overline{J_{\Lambda_i}}$ or $\varphi_i^m < \overline{\varphi_i^m}$ **then** $\mathbb{F} = False, \mathbf{A} = \varnothing$ **break**
7: **else** $\mathbf{A} = \mathbf{A} \bigcup \boldsymbol{config}_i(\mathtt{HI})$
8: **end if**
9: **end for**
10: **else**
11: $\mathbb{F} = False$, $\mathbf{A} = \varnothing$
12: **for** all $P_i \in$ **RP** **do**
13: $p_i(\mathtt{HI}) \leftarrow N_{rp}$
14: calculate constant delay $L_i(\mathtt{HI})$ and jitter $K_i(\mathtt{HI})$
15: simulation to get expected input-output delay L_i^e
16: get the optimal control law for P_i.
17: calculate J_{Λ_i} and φ_i^m
18: **if** $J_{\Lambda_i} \le \overline{J_{\Lambda_i}}$ and $\varphi_i^m \ge \overline{\varphi_i^m}$ **then**
19: $(\mathbb{F}, \mathbf{A}) =$ ControlPriority(**RP** $\setminus P_i, N_{rp} - 1$).
20: **if** $\mathbb{F} == True$ **then break**
21: **end if**
22: **end if**
23: **end for**
24: **end if**

In fact, algorithm 1 solves a decision problem, the maximal times of evaluation (control costs, APM, stability) could be $m + (m-1) + , \cdots, +1 = (m^2+m)/2$. Therefore, the time complexity of algorithm 1 is $O(m^2)$.

5.3 Configuration For Soft Real-Time Tasks

In this section, we discuss how to determine the minimal period scaling factor x and **HI** mode priority assignment for real-time tasks. As the control tasks get shorter periods and higher priorities

in **HI** mode, the periods of soft real-time tasks should be properly extended, otherwise some low priorities tasks may always be preempted by higher priority task and never have a chance to execute. Thus, we propose a scheduablity test in Section 7 to check if a task can complete before its next arrival. The minimal scaling factor x is chosen such that all tasks can pass the scheduablity test.

Since the configuration is computed in offline phase, we can directly search the feasible x in its region. The basic idea is increasing x from 1 to x_{max} with a given step size ε. The maximal feasible x_{max} and ε are given by the designer as a tradeoff between time complexity and quality. Given the factor x, a recursive procedure (discussed in next paragraph) is executed to find the priority assignment for real-time tasks such that the system is schedulable. Once a feasible priority assignment is found, the searching for the minimal x stops and the current x is returned as the minimal period scaling factor x.

The recursive procedure of priority assignment is similar to algorithm 1 except the calculation of control costs and apparent phase margin is replaced by the calculation of response-times, and scheduablity tests take the place of the control quality test (line 6). It's worth noting that soft real-time tasks are assigned priorities from $m+1$ to $m+n$ and control tasks are assigned priorities from 1 to m.

6. STABILITY DURING MODE CHANGE

The stability of plants when MCCS switches from **LO** mode to **HI** mode is discussed in this section. For the switch-back procedure, the problem is simple since MCCS switches back when the plant is at equilibrium point. We first present the formal representation of the closed-loop system, which is a linear system with input.

For each plant P_i, the closed-loop system comprises P_i and a discrete-time LQG controller (the one in **HI** mode). We transform the continuous-time system model (1) into a sampled model. According to [3], the closed-loop system model can be described as[1]:

$$\mathbf{X}_i(k+1) = \mathbf{F}_i\mathbf{X}_i(k) + [I, I]'\mathbf{v}_i(k) + [-\Gamma_i\mathbf{M}_i, -\mathbf{K}_i]'\mathbf{e}_i(k) \quad (15)$$

where $\mathbf{X}_i(k) = [\mathbf{x}_i(k), \tilde{\mathbf{x}}_i(k)]'$, and $\tilde{\mathbf{x}}_i(k)$ denotes the error between actual state $\mathbf{x}_i(k)$ and the estimated state provided by the Kalman filter built-in LQG controller. Constant matrices $\mathbf{F}_i$, Γ_i, $\mathbf{K}_i$, $\mathbf{M}_i$ are calculated based on the plant and controller model. The derivation of these matrices can be found in [3].

The model (15) can be transformed to a linear system with input:

$$\mathbf{X}_i(k+1) = \mathbf{F}_i\mathbf{X}_i(k) + \mathbf{G}\mathbf{U}_i(k) \quad (16)$$

where $\mathbf{G} = \begin{bmatrix} I & -\Gamma_i\mathbf{M}_i \\ I & \mathbf{K}_i \end{bmatrix}$ and $\mathbf{U}_i(k) = [\mathbf{v}_i(k), \mathbf{e}_i(k)]'$.

6.1 Stability Without $\mathbf{U}_i(k)$

The $\mathbf{U}_i(k)$ in (16) comprises noises and environmental disturbances, which hampers the stability. It is necessary to discuss the stability in ideal cases that $\mathbf{U}_i(k) = 0$ before the actual scenarios are considered. Let $\mathbf{F}_i(\mathbf{LO})$ and $\mathbf{F}_i(\mathbf{HI})$ denote the dynamic matrix $\mathbf{F}_i$ of the closed-loop system in **LO** and **HI** mode, respectively. It is worth noting that we assume all the eigenvalues of both $\mathbf{F}_i(\mathbf{LO})$ and $\mathbf{F}_i(\mathbf{HI})$ have magnitudes less than 1, i.e., the system is stable in each mode.

Without $\mathbf{U}_i(k)$, the system (16) is in fact a two-mode switching system, whose stability during mode change can be guaranteed if a common quadratic Lyapunov function (CQLF) can be found.

DEF. 1 (CQLF). *Function $V(\mathbf{x}) = \mathbf{x}^T\mathbf{P}\mathbf{x}$ is a common quadratic Lyapunov function for system (16), if $\mathbf{P}$ is a symmetric and positive definite matrix satisfying:*

$$\mathbf{F}_i(\mathbf{LO})^T\mathbf{P} + \mathbf{P}\mathbf{F}_i(\mathbf{LO}) = -\mathbf{Q}_i(\mathbf{LO}) \quad (17)$$

$$\mathbf{F}_i(\mathbf{HI})^T\mathbf{P} + \mathbf{P}\mathbf{F}_i(\mathbf{HI}) = -\mathbf{Q}_i(\mathbf{HI}) \quad (18)$$

where $\mathbf{Q}_i(\mathbf{LO})$ and $\mathbf{Q}_i(\mathbf{HI})$ are symmetric and positive definite.

There has been lots of work on the necessary and sufficient condition of the existence of a CQLF function [20]. Now, for further analysis, we assume a CQLF of (16) is already found.

6.2 Stability With $\mathbf{U}_i(k)$

Since $\mathbf{U}_i(k)$ is composed of random noise and disturbance, it is impossible to analyze the stability if we do not give any constraints to it. In this paper, the following regulations are adopted.

- The disturbance applied to plant P_i is bound by $\mathbf{db}_i$ while the applied time is no longer than t_i^{db}.
- The interval between two consecutive disturbances is lower bounded by t_i^{itv}.
- The process and the sample noise could be ignored compared to the magnitude of the disturbance.

We analyze the stability of plant P_i under the above admissible inputs based on the stability region. We first introduce the definition of nominal maximal stability region.

DEF. 2 (NOMINAL MAXIMAL STABILITY REGION). *The nominal maximal stability region $NMSR_i$ of plant P_i is defined as the maximal stability region of system:*

$$\mathbf{X}_i(k+1) = \mathbf{F}_i(\mathbf{HI})\mathbf{X}_i(k) \quad (19)$$

Based on Lyapunov's second method, $NMSR_i$ is the set of state $\{\mathbf{X}|\mathbf{X}^T\mathbf{W}\mathbf{X} < 1\}$, where $\mathbf{W}$ can be found by solving the following MAXDET problem [4, 14].

$$\begin{aligned} &\textbf{maximize} && \log\det\mathbf{W}^{-1} \\ &\textbf{subject to} && \mathbf{W} > 0, \textit{and}\ \mathbf{F}_i(\mathbf{HI})^T\mathbf{W}\mathbf{F}_i(\mathbf{HI}) - \mathbf{W} < 0 \end{aligned} \quad (20)$$

The nominal maximal stability region $NMSR_i$ introduced here is the stability region of the **HI** mode application without environment disturbances. Now, we define the stability region of P_i in **HI** mode considering the disturbances.

DEF. 3 (STABILITY REGION). *The stability region (SR_i) of plant P_i is defined as a subset of its $NMSR_i$ so that if its closed-loop system (16) in **HI** mode starts from any state within SR_i with the maximal disturbance, the future states will always stay within the $NMSR_i$.*

According to its definition, the formal representation of SR_i can be given as:

$$SR_i = \{\mathbf{X}_i | f(\mathbf{X}_i, N)^T\mathbf{W}f(\mathbf{X}_i, N) < 1, \forall N \in 1, \cdots, N_i^{db}\} \quad (21)$$

where $f(\mathbf{X}_i, N)$ is the state that N sampling periods later than $\mathbf{X}_i$ under the maximal disturbance:

$$f(\mathbf{X}_i, N) = \mathbf{F}_i^N(\mathbf{HI})\mathbf{X}_i + (\mathbf{F}_i^{N-1}(\mathbf{HI}) + , \cdots, +I)[I, I]'\mathbf{db}_i \quad (22)$$

Integer N_i^{db} is the number of periods of the longest disturbance in **HI** mode, which is

$$N_i^{db} = \lceil t_i^{db}/H_i(\mathbf{LO}) \rceil \quad (23)$$

With the stability region, we discuss how to make plant P_i stable during mode change from **LO** to **HI**. In **LO** mode, the prediction of the next state of (16) is made by the Kalman filter at each period. Then we check if the predicted state $\bar{\mathbf{X}}_i$ stays inside the stability region of plant P_i. If not, stability function Γ_i^s is set to 0 to trigger a mode change to **HI** mode so that the **HI** mode LQG controller can control the plant before it has the chance to be unstable under the maximal disturbance.

[1] See Section 11.4 of [3]

Note that after the disturbance, if the plant is already outside its stability region, it should return to SR_i before the next disturbance is applied. Otherwise the plant may be unstable under two consecutive disturbances. Hence, we have the following constraint:

$$\Re_i(\mathbf{X}_i) \bigcap SR_i \neq \emptyset, \quad \forall \mathbf{X}_i^T \mathbf{W} \mathbf{X}_i = 1 \tag{24}$$

where $\Re_i(\mathbf{X}_i)$ is the reachable set of system (19) starting from state $\mathbf{X}_i$ under no disturbance within t_i^{itv} time units. Denoting M_i^{itv} as $\lceil t_i^{itv}/H_i(\mathtt{HI})\rceil$, we have:

$$\Re_i(\mathbf{X}_i) = \{\mathbf{X}_i(M) | \mathbf{X}_i(M) = \mathbf{F}_i^M(\mathtt{HI})\mathbf{X}_i, M = \{1, \cdots, M_i^{itv}\}\} \tag{25}$$

The stability region and $\Re_i$ can be efficiently calculated in offline manner by using the approach of computing reachable set discussed in [16]. We also claim that by reversely using the constraint (24) and the stability region, we can determine to what extent disturbance can be tolerated by MCCS in **HI** mode.

7. SCHEDULABILITY ANALYSIS

In this section, we present the following schedulability tests for our mixed-criticality system to determine whether it is schedulable in both modes as well as during mode change. We refer the modes before and after mode change as the old mode and new mode, respectively.

(I) schedulability test without mode change (steady mode).

(II) schedulability test of old mode jobs.

(III) schedulability test of new mode jobs.

The first test can be performed easily by using existing response time analysis [18]. The last two tests are adopted to determine if a task is schedulable during mode change. Response-time analyses of all tasks are performed offline in all the tests, considering the interference from the mode change overhead.

For clean demonstration, we show the analysis of the procedure changing from **LO** mode to **HI** mode. The switch-back action follows the similar procedure and can be analyzed similarly. We denote the worst-case response-time of a task τ_i in two steady modes as $R_i(\mathtt{LO})$ and $R_i(\mathtt{HI})$, respectively.

7.1 Test of LO Mode Jobs When Change to HI

Abandoned after the mode change, the old control tasks in **LO** mode can be safely removed from this test. Therefore, we only need to discuss the soft real-time tasks released in **LO** mode in this test. The response time of a **LO** mode task τ_i during mode change can be affected by tasks grouped as:

(A) **LO** mode control tasks having higher priorities.

(B) All **HI** mode control tasks released upon mode change.

(C) Soft real-time tasks only having higher priorities in **LO** mode.

(D) Soft real-time tasks only having higher priorities in **HI** mode.

(E) Soft real-time tasks having higher priorities in both modes.

We suppose the activation of task τ_i occurs z time units before the mode change. A corresponding temporal window $w_i(z)$ is also defined as the interval between the activation and completion of τ_i. Fig. 2 shows a specific example of z and $w_i(z)$.

Now, we discuss the interferences that τ_i receives from the five groups of tasks.

Group A and C First, one can figure out that group A and C both affect task τ_j only before the mode change. Thus the interferences from them are similar in calculation. The interference from a task τ_j in either group can be captured as:

$$F_j^{AC}(z) = \left\lfloor \frac{z}{H_j(\mathtt{LO})} \right\rfloor C_j(\mathtt{LO}) + min\left(z - \left\lfloor \frac{z}{H_j(\mathtt{LO})} \right\rfloor H_j(\mathtt{LO}), C_j(\mathtt{LO})\right) \tag{26}$$

Then, summing them up results in the interferences from both groups:

$$F^{AC}(z) = \sum_{j \in A \bigcup C} F_j^{AC}(z) \tag{27}$$

Group B Tasks in group B are released after mode change, the worst-case interference from this group is:

$$F^B(z) = \sum_{j \in B} \left\lceil \frac{w_i(z) - z}{H_j(\mathtt{HI})} \right\rceil^0 C_j(\mathtt{HI}) \tag{28}$$

where $\lceil a \rceil^0 \triangleq \max(\lceil a \rceil, 0)$.

Group D Tasks in group D only affect τ_i in **HI** mode. Thus, the worst-case interference from this group is:

$$F^D(z) = \sum_{j \in D} \left(C_j(\mathtt{HI}) + \left\lceil \frac{w_i(z) - z - C_j(\mathtt{HI})}{H_j(\mathtt{HI})} \right\rceil^0 C_j(\mathtt{HI}) \right) \tag{29}$$

Group E The interferences from this group last in the whole temporal window $w_i(z)$. Since the WCETs of soft real-time tasks are the same in both modes, the interference can be obtained as:

$$F^E(z) = \sum_{j \in E} \left(\left\lceil \frac{z}{H_j(\mathtt{LO})} \right\rceil + \left\lceil \frac{w_i(z) - z}{H_j(\mathtt{HI})} \right\rceil^0 \right) C_j(\mathtt{HI}) \tag{30}$$

Summing up all the interferences from the five groups and the WCET of itself, the width of the temporal window yields:

$$w_i(z) = C_i(\mathtt{LO}) + F^{AC}(z) + F^B(z) + F^D(z) + F^E(z) \tag{31}$$

This equation can be solved by carrying out a standard recurrence calculation on $w_i(z)$ until the result becomes stable.

Finally, varying z from 0 to the worst-case response time $R_i(\mathtt{LO})$, the worst-case response-time of **LO** mode task τ_i can be obtained as the largest $w_i(z)$:

$$R_i^w = \max(w_i(z)), \forall z \in [0, R_i(\mathtt{LO})). \tag{32}$$

7.2 Test of HI Mode Jobs When Change to HI

We discuss this test with the assumption that previous tests have been passed. In **HI** mode, control tasks are newly released at the mode change and assigned higher priorities than soft real-time tasks. Thus the schedulability of new control task set is not affected and has already been checked in the steady mode tests.

Now we consider the newly released soft real-time tasks in **HI** mode. A **HI** mode task τ_i could be influenced by high-priority tasks grouped as:

(A) All **HI** mode control tasks released upon mode change.

(B) Real-time tasks having higher priorities in both modes.

(C) Real-time tasks only having higher priorities in **HI** mode.

Similar to above section, we use w_i to denote the interval between the activation and completion of τ_i.

Group A Since all control tasks are newly released, the interferences from them could be simply calculated:

$$F^A = \sum_{j \in A} \left\lceil \frac{w_i}{H_j(\mathtt{HI})} \right\rceil C_j(\mathtt{HI}) \tag{33}$$

Group B Task τ_i is released after mode change, indicating its pre-release is finished as its deadline equals period in both modes. Therefore, one can conclude that any task in group B that is released in **LO** mode should complete its execution before the new release of τ_i. The reason is that tasks in this group have higher priorities in both modes, and the completions of their pre-releases certainly require the completions of higher-priority tasks.

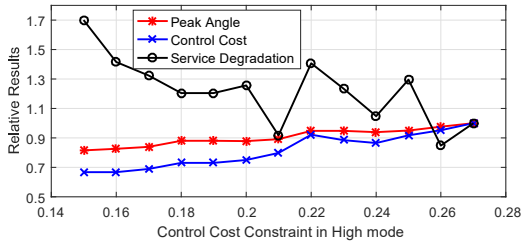

Figure 6: Relative results of the inverted pendulum for different J_{max}. The results when $J_{max} > 0.27$ are not plotted since the system is unstable.

In summary, task τ_i is only affected by newly released tasks in group B and the worst-case interference is:

$$F^B = \sum_{j \in B} \left\lceil \frac{w_i}{H_j(\mathtt{HI})} \right\rceil C_j(\mathtt{HI}) \tag{34}$$

Group C In the worst-case, the interference from a task in group C to task τ_i may contain two parts: the interference from tasks released (a) before mode change and (b) after mode change. Taking both parts into account, the interference from group C yields:

$$F^C = \sum_{j \in C} \left(C_j(\mathtt{HI}) + \left\lceil \frac{w_i - C_j(\mathtt{HI})}{H_j(\mathtt{HI})} \right\rceil^0 C_j(\mathtt{HI}) \right) \tag{35}$$

Again, summing up all the interferences from the three groups and the WCET of itself, we have:

$$w_i = C_i(\mathtt{HI}) + F^A + F^B + F^C \tag{36}$$

Then w_i can be obtained by standard recurrence calculation. Finally the response time for task τ_i which is released in **HI** mode is:

$$R_i^w = \max(w_i, R_i(\mathtt{HI})). \tag{37}$$

8. CASE STUDIES

In this section, we evaluate the effectiveness of our proposed system model (MCCS). All the studies are performed in a computer with an Intel I7 CPU and 16 GB memory.

Firstly, we study the inverted pendulum system presented in the motivation example (Section 3). We implemented our system architecture in MATLAB Simulink combined with the TrueTime toolbox for a time length $T = 20s$. The constraint $\overline{\varphi^m}$ is set as 10. Let J_{max} denote the constraint of the expected control cost in **HI** mode. The simulation result is analyzed in three aspects: (1) The peak angle derivation from vertical θ_p. (2) The local control cost J_{loc} calculated from the definition 12 for $T = 20s$. (3) The service degradation of soft real-time tasks, which is calculated as: $SRV_{de} = (U_n - U_{avg})/U_n \times 100\%$, where U_n is the default utilization and U_{avg} is the average utilization of them in the simulation.

We vary J_{max} from 0.15 to 0.27 with a step of 0.01 and report the results in Fig. 6. Note that all the results have been scaled by the results from scenario $J_{max} = 0.27$ for clear demonstration. The results when $J_{max} = 0.27$ are listed as $\theta_p = 58.5$ degrees, $J_{loc} = 1744$, and $SRV_{de} = 19.0\%$. When $J_{max} \geq 0.27$, the inverted pendulum is unstable after the impulse; therefore, the results are not reported. The **HI** mode periods of the control application $H(\mathtt{HI})$ and the scaling factor x are listed in Table 1.

From Fig. 6 and Tab. 1, we make these observations: (1) As the J_{max} increases, the sampling period becomes larger and the control quality generally worse. When J_{max} is larger than a certain value, the system becomes unstable. This is because the QoC decreases as J_{max} becoming larger. (2) The service of soft real-time tasks is degraded on average about 19% to 32.2% while their periods may be 1.5X to 2.8X longer in **HI** mode. (3) Overall, the degree of service degradation decreases as J_{max} becomes bigger. This is expected since the sampling period of the pendulum becomes larger as J_{max} increases. Therefore, soft real-time tasks can receive

Table 1: **HI** mode periods and scaling factor x for different J_{max}

J_{max}	0.15	0.16	0.17	0.18	0.19	0.20	0.21
$H(\mathtt{HI})(ms)$	1.5	1.7	1.9	2.1	2.1	2.3	2.5
x	2.8	2.3	2.1	1.9	1.9	1.9	1.7
J_{max}	0.22	0.23	0.24	0.25	0.26	0.27	-
$H(\mathtt{HI})(ms)$	2.7	2.9	3.1	3.3	3.5	3.7	-
x	1.7	1.7	1.7	1.7	1.5	1.5	-

more service. However, SRV_{de} is not a strictly decreasing function of J_{max}, for example, when $J_{max} = 0.21$. The reason is that the service degradation is also influenced by the mode change time points chosen by pendulum state and changing conditions. The conditions are unchanged during case studies, but the pendulum state changes dynamically due to sampling and actuation noise.

Next, we compare MCCS with other two approaches in the literature for a set of ten plants. The first approach is called Elastic Task Model (ETM) [6]. This approach gives each task an elastic coefficient and treats task periods as springs. The periods of some tasks can be compressed or decompressed to provide different Quality of Service. The periods of other task can change automatically to prevent the system from overload. The second approach is single-mode approach that is based on a variant of the approach presented in [2]. The approach optimizes the control cost for a set of control tasks and soft real-time tasks under the constraint of system schedulability and robustness requirements of control tasks. We term the second approach OCC.

The set of plants are chosen from [3], including servo, pendulum, motor, water tank and etc. These plants cover a wide range of control properties, such as bandwidth, and thus are considered to be representative. The soft real-time tasks are five periodic tasks whose periods and WCETs are $[40, 30, 30, 55, 65]ms$ and $[6, 3, 2, 6.5, 7]ms$, respectively. The available period sets of the plants are chosen based on the below rules of thumb in [3].

$$0.2 < \omega_b \times h < 0.6$$

where ω_b is the bandwidth of the closed-loop system. We assume the plants have the same elastic coefficients in the the elastic model. We vary the number of plants m from 2 to 10 and compare the results. When $m \in [2, 9]$, ten plants are randomly selected from the set as the evaluated plants. We compare the optimal control cost that the three approaches can provide in the worst cases. We also restrict the apparent phase margin to be positive, that is, all plants must be stable. Table 2 displays the improvements achieved by our approach compared to ETM and OCC in each scenario.

From the results, we make following observations. (1) Compared to OCC, MCCS can improve the control cost about 28.1% on average and up to 38.1% in the 10-plant case. This is expected since the WC-QoC provided by OCC is restricted by soft real-time tasks while in MCCS, the system can relax this limit by temporarily adjusting the periods of soft real-time tasks. (2) Compared to ETM, the control performance in MCCS is 27.4% better on average [2]. The reason is the elastic coefficients in elastic model are fixed parameters. Therefore, when we re-assign the periods of the plant tasks, they should meet the rigid constraints placed by the fixed elastic coefficients, i.e., the periods are adjusted in the same pace. (3) In general, the performance of ETM becomes worse when the number of control tasks increases. This further strengthens the previous observation since the rigidness introduced by the elastic coefficients is amplified when more plants

[2]ETM offers an unstable solution for one task-set in the five-plants scenario. However, to give a feasible result, this solution is not considered when calculating the control performance.

Table 2: Improvement in control cost for scenarios from 2 to 10 plants

m	improves compared to ETM	improves compared to OCC
2	0.6%	12.6%
3	19.4%	25.8%
4	14.4%	29.2%
5	32.6 %	35.4%
6	22.7%	28.6%
7	34.9%	25.4%
8	34.5%	34.7%
9	42.4%	23.4%
10	45.8%	38.1%
average	27.4%	28.1%

are taken in account. In the scenario of two plants, ETM offers a close result with MCCS because the two control tasks can obtain enough resources in both approaches when soft real-time tasks are compressed .

In conclusion, the case study results demonstrates that plants can be guaranteed higher quality of control in MCCS to recover from a critical workspace to a safe workspace at the cost of slight service degradations of low-criticality tasks. Considering the improvement in control quality of much more important control tasks, our proposed system architecture qualifies as a useful scheme.

9. CONCLUSION

In this paper, we present a novel system model for the mixed-criticality system that comprises both control and soft real-time applications. The system can switch from **`LO`** to **`HI`** mode according to the plants' states to ensure the safety and performance in worst-cases, i.e., unexpected disturbances are applied. In **`LO`** mode, the system resources are allocated gracefully between two classes of tasks so that they both receive necessary service. A stability constraint is presented to determine if the system is stable under certain disturbances during mode change. A new approach is also presented to find the optimal system execution configuration in **`HI`** mode so that the **`HI`** mode constraints for control applications are met and the service degradation of soft real-time task is minimized. We implement our system model and scheduling approach to an inverted pendulum and test it with different **`HI`** mode constraints. We also compare our approach with two approaches in the literature for a representative control plant set. Case study results demonstrate the effectiveness of our approach.

10. ACKNOWLEDGEMENTS

This work has been partly funded by China Scholarship Council, German BMBF projects ECU (grant number: 13N11936), and the Fundamental Research Funds for the Central Universities (Grant No: N161604002)

11. REFERENCES

[1] Aminifar and et al. Designing high-quality embedded control systems with guaranteed stability. In *RTSS, 33rd*, pages 283–292. IEEE, 2012.

[2] Amir et al. Aminifar. Control-quality driven design of cyber-physical systems with robustness guarantees. In *DATE*, pages 1093–1098. IEEE, 2013.

[3] Karl J Åström and Björn Wittenmark. *Computer-controlled systems: theory and design*. Prentice Hall, 1996.

[4] Boyd and et al. *Linear matrix inequalities in system and control theory*, volume 15. SIAM, 1994.

[5] Alan Burns and Rob Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[6] Buttazzo and et al. Elastic task model for adaptive rate control. In *Real-Time Systems Symposium*, pages 286–295. IEEE, 1998.

[7] Cervin Anton et al. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on RTCSA*, pages 1–9. Citeseer, 2004.

[8] Liu Xue et al. Ortega: An efficient and flexible online fault tolerance architecture for real-time control systems. *IEEE TII*, 4(4):213–224, 2008.

[9] Goswami and et al. Time-triggered implementations of mixed-criticality automotive software. In *DATE*, pages 1227–1232. EDA, 2012.

[10] Henriksson and et al. Tools for real-time control systems co-design. In *A network for Real-Time research and graduate Education in Sweden*. March 2006.

[11] Robert Hooke and Terry A Jeeves. "direct search"solution of numerical and statistical problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961.

[12] Pratyush et al. Kumar. A hybrid approach to cyber-physical systems verification. In *Proceedings of the 49th DAC*, pages 688–696. ACM, 2012.

[13] Xu-Guang et al. Li. Optimization for networked control systems under the hyper-sampling period. In *ECC*, pages 2868–2873. IEEE, 2014.

[14] Johan Lofberg. Yalmip: A toolbox for modeling and optimization in matlab. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289. IEEE, 2004.

[15] Majumdar and et al. Performance-aware scheduler synthesis for control systems. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 299–308. ACM, 2011.

[16] Oded Maler. Computing reachable sets: An introduction. Technical report, Technical report, 2008.

[17] Oliver Mason and Robert Shorten. On common quadratic lyapunov functions for stable discrete-time lti systems. *IMA Journal of Applied Mathematics*, 69(3):271–283, 2004.

[18] Ola et al. Redell. Exact best-case response time analysis of fixed priority scheduled tasks. In *Real-Time Systems*, pages 165–172. IEEE, 2002.

[19] Soheil et al. Samii. Dynamic scheduling and control-quality optimization of self-triggered control applications. In *RTSS*, pages 95–104. IEEE, 2010.

[20] Robert N Shorten and Kumpati S Narendra. Necessary and sufficient conditions for the existence of a common quadratic lyapunov function for two stable second order linear time-invariant systems. In *American Control Conference, 1999. Proceedings of the 1999*, volume 2, pages 1410–1414. IEEE, 1999.

[21] Wang Tao and et al. Optimal switching law design of periodic switching systems based on mode-dependent average dwell time. In *CCC*, pages 5829–5834. TCCT, 2016.

[22] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.