JMC: Jitter-Based Mixed-Criticality Scheduling for Distributed Real-Time Systems

Kilho Lee, Minsu Kim, Hayeon Kim, Hoon Sung Chwa, *Member, IEEE*, Jaewoo Lee, Jinkyu Lee, *Member, IEEE*, and Insik Shin, *Member, IEEE*

***Abstract*—These days, the term of Internet of Things (IoT) becomes popular to interact and cooperate with individual smart objects, and one of the most critical challenges for IoT is to achieve efficient resource sharing as well as ensure safety-stringent timing constraints. To design such reliable real-time IoT, this paper focuses on the concept of mixed-criticality (MC) introduced to address the low processor utilization on traditional real-time systems. Although different worst-case execution time estimates depending on criticality are proven effective on processor scheduling, the MC concept is not yet mature on distributed systems (such as IoT), especially with end-to-end deadline guarantee. To the best of our knowledge, this paper presents the first attempt to apply the MC concept into interference (or jitter), which is a complicated source of pessimism when analyzing the schedulability of distributed systems. Our goal is to guarantee the end-to-end deadlines of high-criticality flows and minimize the deadline miss ratio of low-criticality flows in distributed systems. To achieve this goal, we introduce a jitter-based MC (JMC) scheduling framework, which supports node-level mode changes in distributed systems. We present an optimal feasibility condition (subject to given schedulability analysis) and two policies to determine jitter-threshold values to achieve the goal in different conditions. Via simulation results for randomly generated workloads, JMC outperforms an existing criticality-monotonic scheme in terms of achieving higher schedulability and fewer deadline misses.**

***Index Terms*—Distributed real-time systems, end-to-end deadline guarantee, jitter-based mixed-criticality (JMC) scheduling, worst-case response time.**

Manuscript received August 29, 2018; revised March 7, 2019; accepted April 24, 2019. Date of publication May 9, 2019; date of current version July 31, 2019. This work was supported in part by the Basic Science Research Program under Grant NRF-2015R1D1A1A01058713, in part by the Engineering Research Center under Grant NRF-2018R1A5A1059921, in part by the Institute for Information & Communications Technology Planning & Evaluation (Resilient Cyber-Physical Systems Research) under Grant 2014-0-00065, and in part by the National Research Foundation under Grant 2015M3A9A7067220, Grant 2019R1A2B5B02001794, Grant 2017H1D8A2031628, Grant 2017M3A9G8084463, and Grant 2018R1C1B5083050. *(Corresponding author: Jinkyu Lee.)*

K. Lee, M. Kim, H. Kim, and I. Shin are with the School of Computing, KAIST, Daejeon 34141, South Korea (e-mail: khlee.cs@kaist.ac.kr; minsu@kaist.ac.kr; hayeon0126@kaist.ac.kr; insik.shin@cs.kaist.ac.kr).

H. S. Chwa is with the Department of Information and Communication Engineering, DGIST, Daegu 42988, South Korea (e-mail: chwahs@dgist.ac.kr).

J. Lee is with the Department of Industrial Security, Chung-Ang University, Seoul 06974, South Korea (e-mail: jaewoolee@cau.ac.kr).

J. Lee is with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea (e-mail: jinkyu.lee@skku.edu).

Digital Object Identifier 10.1109/JIOT.2019.2915790

I. Introduction

NOWADAYS, we have witnessed significant growth in a number of smart objects (or "things") connected to the Internet to interact and cooperate with each other, called the Internet of Things (IoT). Such a trend poses a significant challenge in achieving efficient sharing of computational/communication resources while ensuring safety-stringent timing constraints, primarily to achieve fault isolation/containment, which is also a key to design a *reliable* cyber-physical system, an emerging system of systems often considered as real-time IoT. To provide end-to-end timing guarantees, resource utilization estimates are required for applications; however, conservative worst-case execution time (WCET) estimates have conventionally been used for safety-critical applications, leading to a severely *under-utilized* system in practice. For example, a task typically exhibits a certain variation of execution times depending on the input data and different behavior of the environment. The exact value of WCET is usually unknown and can be overly estimated [1]. A task can also experience a large variation of interference from others depending on scheduling policies and execution scenarios. Though it is feasible to estimate the amount of interference tightly in some environments, it is difficult (often computationally intractable) to calculate it accurately in many complex environments such as distributed systems [2], [3].

In order to narrow such a gap in resource utilization, the concept of mixed-criticality (MC) has been the focus of research in real-time processor(s) scheduling. A key insight [4] is that it provides different levels of timing guarantees to tasks with different criticality levels based on different assurance-levels of parameter estimation. For example, in dual-criticality systems, a substantial number of studies [4], [5] introduce a paradigm where it guarantees the schedulability of both high-criticality (HI) and low-criticality (LO) tasks when the estimation of WCET with low-level assurance is valid. On the other hand, it only guarantees the schedulability of HI tasks when the low assurance-level estimation is violated. In addition to WCET, such a paradigm has been extended toward other parameters, such as period and deadline [6]–[8]. While early studies focused on satisfying the deadlines of HI tasks efficiently, from a practical point of view, there is a growing interest in improving the performance of low-criticality tasks by satisfying their deadlines selectively even in HI mode [6]–[10].

While the concept of MC associated with WCET is proven effective for processor scheduling (in both providing different

levels of timing guarantees and achieving high processor utilization), the MC concept has not matured to support distributed systems, which entails more complicated scheduling with more pessimistic sources. For example, the amount of interference depends on the jitters of higher-priority *flows* in a distributed system (as opposed to *tasks* in a processor), and the pessimism associated with the calculation of worst-case interference for each node is accumulated in the entire system. In this paper, we aim to apply the concept of MC to distributed systems, achieving the following goals for HI and LO flows each of which invokes a series of jobs.

G1: It guarantees that all jobs of HI flows meet end-to-end deadlines.

G2: It maximizes the number of jobs of LO flows that meet end-to-end deadlines.

It entails the following issues to achieve the goals.

I1: What is a key parameter that can be used to capture the pessimistic estimate of interference in distributed systems, while WCET is an effective parameter in processor scheduling?

I2: How should the system react upon detecting the violation of low-level assurance in distributed systems, while the system-wide mode change is typically performed in processor scheduling? For instance, what is the minimum range of nodes to be influenced and how long should such influence last?

I3: How can we estimate interference parameter of low-level assurance in distributed systems, while WCET estimate used as low-level assurance is typically given in processor scheduling?

To address I1, we take advantage of the fact that a distributed system consists of multiple nodes (or stages). End-to-end response time of a flow is calculated as the sum of worst-case response times at each node. Which means, even if a flow experiences more interference than the estimation on some nodes, the overall estimation can still be valid if the other nodes can compensate the exceeded response times. Therefore, it would be reasonable to estimate the interference individually and check its validity at each node upon the completion of execution. To this end, we define *jitter* as the time for individual jobs to reach each node. The jitter at one node directly represents the overall response times up to the previous node; hence it can be used as an indicator for the pessimism contained in the interference estimate at runtime. In this paper, we use jitter to capture the pessimistic estimate of interference in distributed systems.

To address I2, we use the "separation of concerns" principle to efficiently address the complexity of large-scale distributed systems. In distributed systems, there is a ripple effect in which a HI behavior at one node directly (or indirectly) affects interference estimation of other nodes. For example, when a flow receives a larger amount of interference than its estimate at one node, it will reach the next node with a larger jitter value and impose a potentially larger interference on other lower-priority flows on the same node. Since such an interference chain can be long and complicated, it is important to control the ripple effect to the limited region such that it does not spread through the entire distributed system. To this end, we propose jitter-based MC (JMC), an efficient MC scheduling framework for distributed systems, which uses jitters to build the boundary of the ripple effect and control the dependency between nodes. Upon detecting a violation of interference estimate, JMC leverages jitters to enable a mode change on a per-node basis rather than on a system basis. Thereby, JMC minimizes the range of penalized LO flows and also the time duration of HI mode.

While JMC offers an efficient interface for providing different levels of timing guarantees as well as achieving high system utilization, the framework itself does not achieve G1 and G2 without assigning a proper jitter-threshold, a criterion of triggering a node-level mode change. As to I3, we first derive a feasibility condition for the jitter-threshold to accomplish G1 that is optimal subject to given schedulability analysis (e.g., response-time analysis). We then develop two different jitter-threshold assignment policies: the Lazy and Proactive policies. While both satisfy G1 using the feasibility condition, performance of Lazy and Proactive policies depends on the pessimism involved in the analysis.

Note that JMC also follows the essential principle of MC scheduling (i.e., supporting MC tasks in a cost-effective manner); however, the main difference between classic MC and JMC comes from the target system. That is, most existing MC scheduling studies consider a "single" computing node with a *system-wide* mode change according to WCET estimates, but JMC considers "multiple" resources with a *stage-level* mode change according to *jitter* values, by answering I1–I3. More detailed explanation of the difference will be presented at the end of Section IV.

To evaluate the effectiveness of our JMC scheme compared to existing approaches such as criticality-aware policies [e.g., criticality-aware deadline monotonic (CA-DM)], we conduct simulations with randomly generated flows. Simulation results show that JMC outperforms existing criticality-aware scheduling in terms of achieving both G1 and G2.

In summary, this paper makes the following contributions.

1) We present the first study to apply the MC concept to address the pessimism in the worst-case interference analysis of *distributed systems*, enabling mode changes on a per-node basis to minimize penalty of LO flows.
2) We propose JMC, a framework that offers an efficient jitter-based interface for providing different levels of timing guarantees as well as achieving high system utilization.
3) We develop jitter-threshold assignment policies that JMC leverages to achieve G1 and G2.
4) Our simulation results show that JMC is effective in achieving higher schedulability and lower deadline miss ratio of LO flows.

The rest of this paper is structured as follows. We present our system model in Section II, and explain the motivation of this paper in Section III—why jitter-based mode change is needed. Following Sections IV and V, we develop a JMC scheduling framework for distributed systems. In Section VI, we propose two jitter-threshold assignment policies that achieve G1 and G2. Evaluation of JMC associated with the jitter-threshold assignment policies is presented in

Section VII, and we discuss a possible extension of JMC in Section VIII. Section IX summarizes related work, and Section X ends this paper with a conclusion.

II. System Model

A. Target System

We consider an MC distributed system which consists of multiple computing nodes connected via network links. In a computing node, tasks are executed, whereas in a network link, messages are transmitted. An *end-to-end flow* (*flow* in short)[1] is a set of tasks that need to be executed in the same or different computing nodes, while transmitting messages in between. As system complexity increases, many IoT systems have been designed in this distributed architecture. For instance, a smart factory system may generate an end-to-end flow (consisting of a series of tasks) which collects raw data from sensors, processes the data with some filters, determines control inputs based on the data, and feeds the control inputs to actuators. Each task runs on different nodes and forwards its result to the next node via network links. In addition, for simplicity, we restrict our attention to *dual-criticality* systems where there are two criticality levels: 1) HI (high-criticality) and 2) LO (low-criticality), as in many mixed-critical scheduling studies [5]–[10], [12]–[14]. Flows are given different criticality according to the severity of a deadline miss: HI and LO given to more and less significant flows, respectively.

B. Task Model

We consider a sporadic distributed task model $\mathbf{\Gamma}$, where each flow $\Gamma_i \in \mathbf{\Gamma}$ can be released aperiodically with a minimum interarrival time T_i. A flow Γ_i consists of a series of n_i *steps* (tasks or messages) that are either executed or transmitted on *stages* (nodes or links) [15]. The first step of each flow is released by a sporadic external event on the first stage. Then, every following step is released after the corresponding preceding step is completed. The kth step of a flow Γ_i is denoted as $\tau_{i,k}$. While a step $\tau_{i,k}$ denotes a task or message, we refer a resource (a node or link) as a stage, $s_{i,k}$ or s_m; the former indicates the kth stage of Γ_i (i.e., the link or node on which $\tau_{i,k}$ is executed or transmitted) and the latter represents the stage with the global index m. Also, we denote the set of stages of flow Γ_i as $S_i = \{s_{i,1}, \ldots, s_{i,n_i}\}$. We call an instance of a flow as a *job*. We define Γ_i^j and $\tau_{i,k}^j$ as the jth job of Γ_i and $\tau_{i,k}$, respectively. The relative deadline of a flow Γ_i is represented by D_i. Here, we assume the deadline is not larger than T_i (i.e., $D_i \leq T_i$). Also, L_i is a criticality level of Γ_i (i.e., HI or LO). Note that the criticality level is an inherent attribute of each flow; it is determined and fixed at design time, according to the severity of a deadline miss. With the criticality level, we use $\Gamma(\text{HI})$ to denote a set of HI flows, and $\Gamma(\text{LO})$ a set of LO flows. For each step $\tau_{i,j}$, we define $C_{i,j}$ as the maximum required time spent on stage $s_{i,j}$, that is, the WCET on $s_{i,j}$ when it is a computing node, or the worst-case transmission time on $s_{i,j}$ when it is a network link. Note that while the transmission time at each link may also vary depending on the data size sent by its preceding node, we consider the worst-case transmission time as in the WCET.

[1]We follow the definition of MARTE [11], the OMG standard specification widely used in distributed real-time systems.

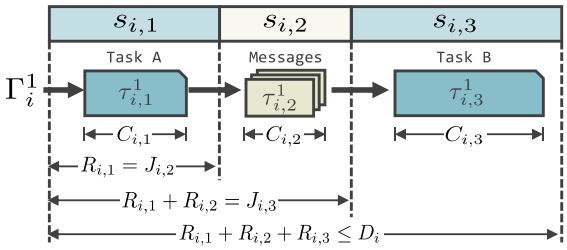


Fig. 1. System model overview: the flow Γ_i goes through three stages, composed of two computing nodes and one communication links.

Fig. 1 depicts an example of a flow Γ_i that goes through two computation nodes and one network link. An external event instantiates Γ_i, producing the first job in the system. Then, a series of steps ($\tau_{i,1}^1$, $\tau_{i,2}^1$, and $\tau_{i,3}^1$) is released sequentially, each on corresponding stages ($s_{i,1}$, $s_{i,2}$, and $s_{i,3}$). On $s_{i,1}$ and $s_{i,3}$, which are computing nodes, tasks $\tau_{i,1}^1$ and $\tau_{i,3}^1$ are executed, respectively. On $s_{i,2}$, which is a network link, a message $\tau_{i,2}^1$ is transmitted.

We assume the values of T_i, $C_{i,k}$, and D_i for each flow are given, which remain unchanged while the system is running. We would like to emphasize that this paper newly proposes MC scheduling based on the threshold for *jitter*, not that for the WCET. Therefore, the task model considers a single parameter of $C_{i,k}$, rather than both $C_{i,k}(\text{LO})$ and $C_{i,k}(\text{HI})$. After addressing the JMC scheduling clearly, we will discuss how to incorporate both thresholds for jitter *and* the WCET into MC scheduling in Section VIII.

C. Network Model

We consider point-to-point network links between computing nodes. On each link, a message can be divided into multiple *packets*, depending on the maximum transmission unit (MTU) of the link (e.g., 1500 bytes on Ethernet). Each link forwards packets based on the assigned priorities of packets; the higher priority packets are forwarded ahead of the lower priority ones. In addition, since each packet is nonpreemptive, messages can be blocked by at most one nonpreemptible lower-priority packet.

D. Scheduling Algorithm

We consider fixed-priority scheduling for each stage, and assume flow priorities are distinct. We consider two priority assignment schemes: 1) deadline-monotonic (DM) and 2) CA-DM. Under DM, all HI and LO flow priorities are ordered according to relative end-to-end deadlines (i.e., D_i) without distinction; the shorter relative end-to-end deadline gets assigned with the higher priority. Under CA-DM, flows are first sorted according to the criticality, and then priority is assigned depending on relative end-to-end deadlines, such that all HI flows get higher priorities than all LO flows.

E. Parameters

$J_{i,k}$ is the *release jitter* of $\tau_{i,k}$ on stage $s_{i,k}$, which is defined as the time duration between the release of $\tau_{i,1}$ (initial release)

and the arrival of $\tau_{i,k}$; by definition, $J_{i,1} = 0$. $J_{i,k}$ is determined at runtime depending on how much interference the job experiences by other higher priority flows. We introduce an additional parameter $J^*_{i,k}$ that is the upper-bound of $J_{i,k}$.[2]

III. Background and Motivation

As mentioned in the introduction, our goal is to provide end-to-end timing guarantees for *every* HI flows (i.e., achieving G1) and *as many* LO flows *as possible* (i.e., achieving G2). One of the typical ways to achieve G1 and G2 is to calculate the worst-case response-time (WCRT); as long as WCRT for a target flow Γ_i is not larger than its relative deadline, any job of the flow never misses its deadline. Although the concept of WCRT provides an intuitive interface for end-to-end timing guarantees, its efficiency completely depends on how pessimistically WCRT is calculated. In this section, we first present a typical WCRT analysis method. We then investigate the reason and source of the pessimism in the analysis, which is the main obstacle for achieving G1 and G2 at the same time.

A. Response Time Analysis

To provide end-to-end timing guarantees, there have been many analysis methods that deal with end-to-end response time for distributed systems, including holistic analysis [16], real-time calculus (RTC) [17], offset-based approach [18], and compositional performance analysis [19]. For the sake of clarity, we focus on the holistic analysis on fixed-priority scheduling, throughout this paper.

Holistic analysis is developed to find the worst-case response time (WCRT) of a distributed flow set under fixed-priority preemptive scheduling. Let $R^*_{i,k}$ denote the WCRT of step $\tau_{i,k}$, and R^*_i denote the end-to-end WCRT of Γ_i. In other words, $R^*_{i,k}$ is an upper-bound of the duration of every job of Γ_i between its arrival and completion on $\tau_{i,k}$, and R^*_i is an upper-bound of the duration of every job of Γ_i between its release on the first step $\tau_{i,1}$ and its completion on the last step τ_{i,n_i}.

$R^*_{i,k}$ occurs after a critical instant, when: 1) Γ_i and all higher-priority flows Γ_j arrive at $s_{i,k}$ at the same time while 2) the jobs of each Γ_j arrive at the minimum interarrival time after experiencing the maximum possible jitter [20]. $R^*_{i,k}$ can be calculated by the following fixed-point iteration [16], [21]:

$$R^{*\,(n+1)}_{i,k} = C_{i,k} + B_{i,k} + \sum_{\tau_{j,k'} \in hp(i,k)} \left\lceil \frac{R^{*\,(n)}_{i,k} + J^*_{j,k'}}{T_j} \right\rceil C_{j,k'} \quad (1)$$

where $hp(i,k)$ denotes a set of steps of higher-priority flows Γ_j that execute on stage $s_{i,k}$ while sharing the resource; recall that $\tau_{j,k'}$ represents the k'th step of the flow Γ_j. $B_{i,k}$ denotes the longest time that $\tau_{i,k}$ is blocked by a lower priority flow. Considering a stage for a computing node and that for a communication link employ preemptive and nonpreemptive scheduling, respectively, $B_{i,k}$ for the former is zero and the latter is the transmission time of a single packet having the size of MTU.

[2]In this paper, we put "*" to distinguish the actual value of the parameter at runtime (i.e., $J_{i,k}$) and an upper-bound on the value (i.e., $J^*_{i,k}$).

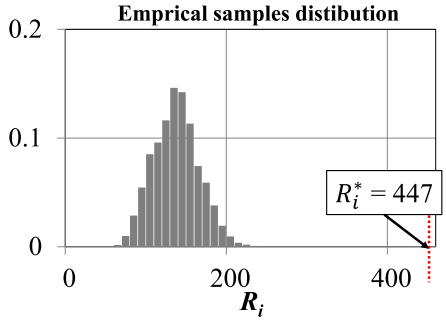


Fig. 2. Pessimism in calculating interference in the worst-case response time analysis: the graph represents distribution of actual response times and R^*_i is a worst-case analytical bound calculated as 447 from the holistic analysis.

The iteration starts with $R^{*\,(0)}_{i,k} = C_{i,k}$, and ends when $R^{*\,(n)}_{i,k} = R^{*\,(n+1)}_{i,k}$ or $R^{*\,(n)}_{i,k} > D_i$ (deemed unschedulable). The maximum possible jitter $J^*_{j,k'}$ then can be computed as follows:

$$J^*_{j,k'} = \sum_{m=1}^{k'-1} R^*_{j,m}. \quad (2)$$

Also, R^*_i can be computed as the sum of $R^*_{i,k}$ as follows:

$$R^*_i = \sum_{s_{i,k} \in S_i} R^*_{i,k}. \quad (3)$$

Thus, it is guaranteed that Γ_i meets all end-to-end deadlines if $R^*_i \le D_i$.

B. Pessimism in WCRT Analysis

1) Motivational Simulation: The holistic WCRT analysis (presented in Section III-A) is useful to investigate whether each individual flow Γ_i can satisfy all timing constraints. However, such WCRT analysis is often pessimistic, in particular, for distributed task models. As an example, Fig. 2 illustrates a typical pessimism involved in the WCRT calculation. For a given flow Γ_i, the figure shows an analytical WCRT bound (i.e., R^*_i) that is calculated according to the holistic analysis and distribution of actual response time measurements that are obtained through simulation.[3] The figure indicates a substantially large gap between the actual response times and its analytic upper-bound (i.e., WCRT); the former ranges between 60 and 250 while the latter is 447, which is too much large. Such a pessimistic analysis inevitably leads to low resource utilization and becomes a critical reason for performance degradation.

2) Sources of Pessimism: One may wonder how often such pessimism can happen and how serious it can be. To answer them, it is necessary to understand the root causes of pessimism in the holistic WCRT analysis.

[3]We generated Γ_i that goes through three computing nodes and two communication links with T_i of 500 and $C_{i,k}$ of 10 (for all k). In each computing node, Γ_i was interfered by six higher priority flows Γ_j which have T_j and $C_{j,k}$ randomly drawn from [10, 100] and $[0.05 * T_j/n_j,\ 0.15 * T_j/n_j]$, respectively, where n_j denotes the number of steps of Γ_j. Note that in each node, three out of six higher priority flows went through one stage; other three flows went through five stages, so that they showed fluctuating interarrival time between their jobs.

Note that the response time of a flow Γ_i is simply the sum of the execution time of Γ_i and the total amount of *interference* imposed by other higher-priority flows. When $C_{i,k}$ is given, pessimism in the WCRT analysis mainly arises from overestimation of interference, due to the extreme assumptions that the analysis inherently possesses for strict timing guarantees. As explained in Section III-A, $R^*_{i,k}$ is computed under certain assumptions, where: a) the jobs of each higher-priority flows Γ_j *arrive at the minimum interarrival time* after experiencing the maximum possible jitter; b) Γ_i and all higher-priority flows Γ_j *arrive at* $s_{i,k}$ *at the same time*; and then c) R^*_i is computed as the *sum of* $R^*_{i,k}$. We identify each source in more detail.

3) Per-Flow Minimum Interarrival Time: The first pessimism in the calculation of $R^*_{i,k}$ is associated with the analysis of minimum interarrival time. One of the necessary conditions for the maximum interference of a higher-priority flow Γ_j imposed to a lower-priority flow Γ_i is that the consecutive jobs of Γ_j arrive with the minimum possible interarrival time. This happens when: a) the first job of Γ_j arrives with the maximum possible jitter, after experiencing a worst-case interference on every previous stage [see (2)] and b) the second and subsequent jobs of Γ_j arrive with the minimum possible jitters. However, such condition occurs with a very low probability resulting in an overestimation of the interference.

4) Per-Stage Critical Instant: Another pessimism in the calculation $R^*_{i,k}$ is due to the assumption on the critical instant. For stage $s_{i,k}$, $R^*_{i,k}$ is computed assuming that all higher priority jobs arrive at $s_{i,k}$ simultaneously after experiencing their own maximum jitters. However, such condition is only a small fraction of large arrival combinations that higher-priority flows can create collectively. The probability of a critical instant exponentially decreases in accordance with the number of nonharmonic period flows.

5) End-to-End Delays: The calculation of the end-to-end R^*_i involves additional pessimism that all the per-flow and per-stage pessimism mentioned in the above occurs on *every stage*. According to (3), R^*_i is computed as the *sum of* $R^*_{i,k}$, which accumulates above mentioned pessimism throughout the passing stages.

IV. Approach Overview for JMC

Motivated by the pessimism of WCRT explained in Section III, this section presents an approach overview of JMC, an efficient JMC scheduling framework that achieves G1 and G2 in distributed systems. To this end, we design the concept of stage-level mode change based on introducing a new jitter-threshold parameter. We then show an example that demonstrates the effectiveness of the new design in achieving G1 and G2. Finally, we define new notions necessary for realizing the design to be used for Section V.

Pessimistic WCRT analysis inherently hinders our goal of maximizing the number of LO jobs that meet deadlines subject to satisfying the deadlines of all HI jobs. We can consider two approaches to address the pessimism involved in the analysis. One is to develop a tighter WCRT analysis, and a number of studies have been made in this direction, e.g., [18], [20], and [22]. The other is to employ an optimistic WCRT estimate and to construct a mode-based scheduling that provides a different type of guarantees per mode. The latter is a new direction proposed in this paper, which has been little studied and is orthogonal to the former approach.

The key idea of our mode-based scheduling is to apply different scheduling strategies in different modes according to the optimistic WCRT estimates. We first introduce an optimistic WCRT estimate, denoted as R^o_i, of Γ_i (i.e., $R^o_i < R^*_i$). The value of R^o_i can be used as a guideline for determining the criticality mode of each stage at runtime.[4] In particular, all stages (i.e., nodes and links) start in LO mode during which all flows are assigned resources based on their optimistic WCRT estimates and scheduled together to achieve both G1 and G2. However, once any HI flow experiences a larger response time than its optimistic WCRT estimate, stages associated with the HI flow switch to HI mode during which HI flows get strictly higher priority than LO ones to still achieve both G1 and G2 with their pessimistic WCRT estimates.

Another challenge is to properly handle HI behavior without compromising requirements of other flows in large-scale distributed systems. In distributed systems, there is a ripple effect in which HI behavior at one stage directly (or indirectly) affects interference estimation of other stages. To handle HI behavior while minimizing the range of nodes to be influenced by HI behavior, we also introduce per-stage optimistic WCRT estimate. Since Γ_i goes through a series of stages $s_{i,k}$, its optimistic WCRT estimate R^o_i can be split into per-stage optimistic WCRT estimates $\{R^o_{i,k}\}$, and we can detect a violation of R^o_i separately on each stage. This can be done since the end-to-end response time R_i is computed as the sum of individual response times on each stage $R_{i,k}$. It enables the following: even if a flow experiences more interference than the estimation on a certain stage (i.e., $R_{i,k} > R^o_{i,k}$) at runtime, the overall estimation can still be valid if the rest of the stages can compensate for the exceeded response time. Therefore, we introduce the concept of *stage-level mode change* under which we check the validity upon the completion of execution on each stage $s_{i,k}$ by checking the sum of response times up to that stage and perform mode change at each stage. Here, each stage has its own criticality mode (HI or LO) that indicates what scheduling policy to be applied in response to runtime behavior of flows on each stage; it effectively limits the influence of HI behavior to the system. In contrast to classical MC scheduling, in JMC, the notion of criticality (mode) no longer represents the criticality of the entire system. Instead, it represents the criticality of each stage.

For the stage-level mode change, we check whether $\sum_{m=1}^{k} R_{i,m} \leq \sum_{m=1}^{k} R^o_{i,m}$ holds on each stage. If it is violated, we change the mode of the *next stage* $s_{i,k+1}$ to HI. As to simplify the notation, our mode-based scheduling framework uses the *release jitter* $J_{i,k}$ to effectively detect the violation of optimistic WCRT estimate. As shown in Fig. 1, $J_{i,k}$ directly represents the sum of response times up to the previous stage $s_{i,k-1}$ [see (2)]. That is, we denote $J^o_{i,k}$ as an optimistic release jitter on stage $s_{i,k}$ and define $J^o_{i,k}$ as $\sum_{m=1}^{k-1} R^o_{i,m}$. This way,

[4] In this section, we assume that the value of R^o_i is given, and we will discuss issues regarding how to determine the appropriate value later in Section VI.

TABLE I
EXAMPLE PARAMETERS

Flows	T_i	C_{i,n_i}	D_i	L_i
Γ_1	10	3	9	HI
Γ_2	7	3	4	LO

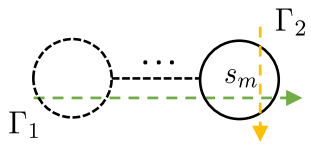


Fig. 3. Example topology.

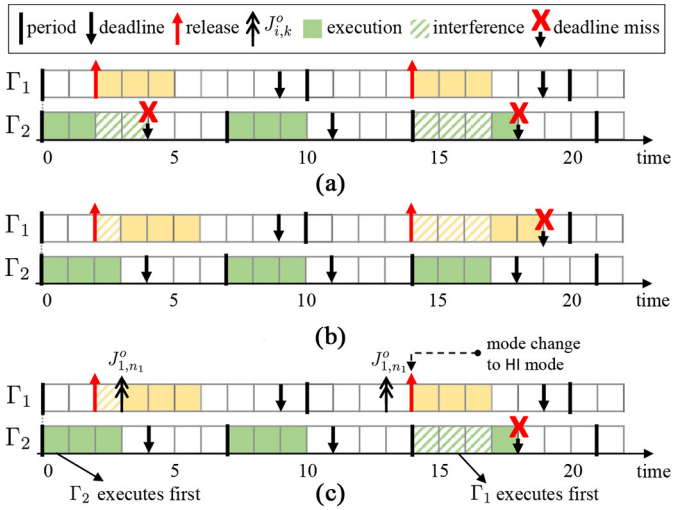


Fig. 4. Scheduling of the example. (a) CA-DM scheduling. (b) EDF scheduling. (c) JMC scheduling.

our framework can effectively check the violation of an optimistic jitter estimate (also called jitter-threshold) on each stage $s_{i,k}$ (i.e., $J_{i,k} > J^o_{i,k}$) upon arrival of a job, and perform per-stage mode change so as to satisfy the timing constraints of HI flows. In summary, our framework maximizes the system performance (in terms of satisfying the deadlines of LO flows) as well as satisfy the deadlines of all HI flows.

Example: In order to illustrate the benefit of our mode-based scheduling, we present a motivational example with two flows; Γ_1 and Γ_2 (see Table I). We assume that Γ_1 goes through a series of stages and have contention on stage s_m with Γ_2, as shown in Fig. 3. Γ_2 is a flow that executes on a single stage s_m. Γ_1 is a HI flow with its period (T_1) of 10, and the execution time on stage s_m is 3. Γ_2 is a LO flow with its period (T_2) of 7, and the execution time on stage s_m is 3. Each flow's deadline (D_i) is 9 and 4, respectively.

Fig. 4 shows the scheduling on stage s_m under various scheduling strategies: CA-DM, EDF, and JMC. Jobs of Γ_1 arrive at stage s_m at 2 and 14, and jobs of Γ_2 are released at stage s_m at 0, 7, and 14. We first consider a case where CA-DM scheduling algorithm is used. Since CA-DM favors HI flows, it always assigns a higher priority to Γ_1 than Γ_2. Therefore, WCRT of Γ_2 is calculated as $R^*_2 = 6$ according to the WCRT analysis. This leads to $R^*_2 > D_2$, which means that Γ_2 fails the schedulability test of CA-DM. It is shown in the figure that two jobs (the first and the third) of Γ_2 miss deadlines, which does not help achieve our goal G2. On the other hand, EDF scheduling, which is an optimal scheduler for single-criticality systems, can reduce the number of deadline misses by assigning higher priorities to the jobs with earlier deadlines regardless of the criticality of flows. However, it may result in deadline misses of HI flows (i.e., the second job of Γ_1), which is against our goal G1. Note that both CA-DM and EDF apply a *single* scheduling policy that does not change at runtime.

Our framework JMC, on the other hand, employs different scheduling policies depending on the mode of each stage. In Fig. 4, the optimistic release jitter J^o_{1,n_1} is 3; we will explain how to determine the value of $J^o_{i,k}$ in Section VI. Upon arrival of a job on stage s_m, actual release jitter J_{1,n_1} is compared with J^o_{1,n_1} to decide the operating mode of the stage. In the example, the first job of Γ_1 satisfies $J_{1,n_1} \leq J^o_{1,n_1}$ $(2 \leq 3)$, thus s_i is in LO mode. In LO mode, flows are scheduled with DM, each of which assigns its priority according to its deadline only, regardless of its criticality. On the other hand, the second job of Γ_2 arrives later than its release time plus J^o_{1,n_1} (i.e., $J_{1,n_1} > J^o_{1,n_1}$), thus triggering the mode change of s_m to HI mode. In HI mode, flows are scheduled according to CA-DM, which strictly gives higher priority to HI flows. Note that if we do not change the mode of s_m to HI mode, Γ_1 misses its deadline. This way, JMC is able to achieve both G1 and G2.

To enable the design principles of JMC explained so far, we define the following notions associated with a jitter $J_{i,k}$ and its optimistic threshold $J^o_{i,k}$.

Job Behavior: A job $\tau^j_{i,k}$ is said to *exhibit LO behavior* on a stage $s_{i,k}$ if it arrives at $s_{i,k}$ within its optimistic jitter threshold ($J_{i,k} \leq J^o_{i,k}$). A job is said to *exhibit HI behavior* otherwise, i.e., if it takes more than $J^o_{i,k}$ time units for $\tau^j_{i,k}$ to reach $s_{i,k}$ after the release of $s_{i,1}$.

Stage Mode: Let $Z_{i,k}(t)$ denote a set of jobs $\tau^j_{i,k}$ that arrived at $s_{i,k}$ but are not yet completed at time t. A stage $s_{i,k}$ is said to be in *LO mode* at time t if for every job $\tau^j_{i,k} \in Z_{i,k}(t)$, $\tau^j_{i,k}$ exhibits LO behavior. The stage $s_{i,k}$ is said to be in *HI mode* at time t otherwise, i.e., if there exist one or more jobs $\tau^j_{i,k} \in Z_{i,k}(t)$ that exhibits HI behavior.

JMC-Schedulability: An MC distributed system $\mathbf{\Gamma}$ is defined to be *JMC-schedulable* by a scheduling algorithm if the following two conditions hold.

1) Every job invoked by a HI flow in Γ(HI) meets an end-to-end deadline.
2) Every job invoked by a LO flow in Γ(LO) meets an end-to-end deadline if it passes only LO mode stages.

Difference Between JMC and Classic MC: We would like to emphasize that the notion of the JMC scheduling is totally new, and explain the difference between JMC and classic MC as follows. In MC scheduling, the mode-based scheduling is an essential principle to support MC tasks in a cost-effective manner. It employs multiple estimates for a certain parameter under each mode [e.g., C(LO) and C(HI) for the WCET], and changes the mode when runtime behavior for the parameter violates the estimate.

While JMC also follows the principle of MC scheduling, the difference between JMC and the classical MC comes from the target system. Most MC scheduling studies employ the WCET parameter (C) to determine the mode, since it is effective to alleviate pessimistic resource reservation on a single node where timing guarantee for a job is determined by the order of execution. In addition, the most MC scheduling studies trigger the system-wide mode change (i.e., once a mode change occurs, it applies to all the jobs in the system) on a single node, since a violation of C(LO) by a particular job directly affects timing guarantees of all other (lower-priority) jobs in the system. On the other hand, if we focus on a distributed system, timing guarantee for a flow (corresponding to a job in a single node system) is determined by the order of execution on all the resources (i.e., computing nodes and network links) that the flow uses. This entails selecting not only a proper parameter that triggers a mode change, but also a new coverage of the distributed system in which a mode change applies; note that both parameter and coverage to be determined should be able to alleviate pessimistic resource reservation on "multiple" resources. In addition, it is important to assign a proper jitter-threshold value for each stage so as to achieve G1 and G2.

In summary, JMC has the following distinctions with classic MC scheduling.

1) JMC uses the jitter (J) parameter to determine the mode.
2) JMC proposes a stage-level mode change, while distinguishing the minimum range of nodes to be influenced.
3) JMC proposes a strategy to find out a proper jitter-threshold for each stage.

Although this paper mainly focuses on MC scheduling with the jitter parameter, JMC can be extended with the other parameters. For instance, JMC is orthogonal to the execution time-based classic MC scheduling (see Section VIII for more details).

V. Jitter-Based MC Scheduling Framework

Based on the approach overview in Section IV, this section develops JMC, which efficiently reduces the pessimism of WCRT estimates. A key feature of JMC is performing a stage-level mode change when detecting a HI behavior of a job on a stage, instead of system-wide or path-wide mode change. In this way, JMC penalizes a minimal set of LO flows to guarantee the schedulability of HI flows upon a violation of WCRT estimate assumption. It is also important to perform a HI-to-LO mode change as soon as possible to maximize the performance of LO flows. Since JMC takes a divide-and-conquer approach in a sense that it conducts mode change on individual stages, it is possible to change the mode of a stage back to LO as soon as the job exhibiting HI behavior leaves the stage. Now, we first detail the mode change protocol and scheduling for JMC, and then analyze JMC including the range of $J^o_{i,k}$ that guarantees the JMC-schedulability.

A. Mode Change Protocol

Algorithm 1 presents the stage-level mode change protocol. Each stage in the system starts in LO mode. If any job of HI flow Γ_i exhibits a HI behavior on stage $s_m = s_{i,k}$, we change the mode of the stage s_m only, instead of carrying out the system-wide mode change. That is, when a job of Γ_i with $L_i =$ HI arrives at s_m, we compare $J_{i,k}$ and $J^o_{i,k}$. If $J_{i,k} > J^o_{i,k}$ (i.e., a job of a HI flow does not arrive at stage $s_{i,k}$ within the $J^o_{i,k}$ estimate), we set the mode of s_m to HI. The mode of s_m is changed back to LO when all the jobs exhibiting HI behavior on stage s_m complete execution (or transmission). To this end, we insert the index of the job exhibiting HI behavior into σ_m (note that when the system starts, σ_m is set to $\emptyset$ for each s_m). If the job finishes its execution on s_m, we remove the index of the job from σ_m; if σ_m becomes empty, we change the mode of s_m back to LO. On the other hand, we simply drop a job of LO flow Γ_i if it violates $J^o_{i,k}$ estimation. This isolates the effect of such a LO job from HI jobs in other stages.

Algorithm 1 Stage-Level Mode Change Protocol

When a job $\tau^j_{i,k}$ arrives at s_m (=$s_{i,k}$),

```
if J_{i,k} > J^o_{i,k} then
    if L_i = LO then
        DROP τ^j_{i,k}
    else
        if s_m is in a LO mode then
            change the mode of s_m to HI
        end if
        σ_m ← σ_m ∪ {τ^j_{i,k}}
    end if
end if
```

When $\tau^j_{i,k} \in \sigma_m$ finishes its execution

```
σ_m ← σ_m \ {τ^j_{i,k}}
if σ_m = ∅ then
    change the mode of s_m to LO
end if
```

B. Mode Scheduling

Each stage employs different fixed-priority scheduling policies in HI and LO modes as long as it preserves the following priority relationship: for each pair of two flows of the same criticality level, the priority relationship between the two flows must stay consistent in both HI and LO modes. That is, for all $\Gamma_i \in \Gamma$(HI), Γ_i should be of a higher priority than Γ_j in HI mode if the same relationship holds in LO mode, and so is the case with LO flows.

To this end, we consider DM in LO mode and CA-DM in HI mode. CA-DM prioritizes flows in two-steps: first, according to criticality (HI flows first), and then according to deadline (shorter relative deadline first). This way, the priority relationship between two flows of the same criticality level remains the same regardless of HI and LO modes. Note that JMC can be applied with any other flow-level fixed priority assignment algorithms. In Section VII, we show how well JMC works when it combines with various priority assignment algorithms.

C. Response Time Analysis

We first introduce some notations and convention to analyze JMC. Let $R_{i,k}$ denote the actual response time of a job of step $\tau_{i,k}$ on stage $s_{i,k}$ regardless of the mode at $s_{i,k}$. $R_{i,k}$(HI)

[*likewise*, $R_{i,k}(\mathsf{LO})$] indicates $R_{i,k}$ in the case where $s_{i,k}$ is HI (*likewise*, LO) mode when Γ_i reaches $s_{i,k}$. We will use $*$ to denote an upper bound. For example, $R^*_{i,k}(\mathsf{HI})$ and $R^*_{i,k}(\mathsf{LO})$ are the upper bounds of $R_{i,k}(\mathsf{HI})$ and $R_{i,k}(\mathsf{LO})$, respectively.

We can compute $R^*_{i,k}(\mathsf{HI})$ and $R^*_{i,k}(\mathsf{LO})$ using (1) while replacing $hp(i,k)$ with $hp(i,k,\mathsf{HI})$ and $hp(i,k,\mathsf{LO})$, respectively, since Γ_i has a different set of higher-priority flows under different scheduling algorithms in different modes. For instance, $hp(i,k,\mathsf{LO})$ will be defined according to DM in LO mode and $hp(i,k,\mathsf{HI})$ according to CA-DM in HI mode.

Every HI flow $\Gamma_i \in \Gamma(\mathsf{HI})$ satisfies the following three properties. First, $R^*_{i,k}(\mathsf{HI}) \leq R^*_{i,k}(\mathsf{LO})$; this is because $hp(i,k,\mathsf{HI}) \subseteq hp(i,k,\mathsf{LO})$ holds, where $hp(i,k,\mathsf{LO})$ contains LO flows but $hp(i,k,\mathsf{HI})$ does not. Second, when $\tau_{i,k}$ experiences a mode change (i.e., LO to HI) in the middle of execution, $R_{i,k} \leq R^*_{i,k}(\mathsf{LO})$; this is because all LO flows no longer interfere $\tau_{i,k}$ after the mode change. Third, when $\tau_{i,k}$ exhibits HI behavior on $s_{i,k}$, $R_{i,k} \leq R^*_{i,k}(\mathsf{HI})$; this is because the mode of $s_{i,k}$ will be changed to HI upon the arrival of $\tau_{i,k}$, so that $\tau_{i,k}$ will be interfered by HI flows only.

In addition, we will use R_i to denote the actual end-to-end response time of a job in flow Γ_i for execution on its whole stages S_i regardless of the mode in each individual stage. We let $R_i(\mathsf{HI})$ and $R_i(\mathsf{LO})$ denote the response times of a job of Γ_i when all the stages it passes exhibit HI and LO mode, respectively, and $R^*_i(\mathsf{HI})$ and $R^*_i(\mathsf{LO})$ denote an upper-bound of $R_i(\mathsf{HI})$ and $R_i(\mathsf{LO})$, respectively.

Necessary Condition: We present a necessary condition for the scheduling scheme of JMC in the following lemma.

Lemma 1: A necessary condition for the JMC-schedulability under JMC is

$$\forall \Gamma_i \in \Gamma(\mathsf{HI}), R^*_i(\mathsf{HI}) \leq D_i \text{ and } \forall \Gamma_j \in \Gamma(\mathsf{LO}), R^*_j(\mathsf{LO}) \leq D_j. \tag{4}$$

Proof: It is trivial to see that it will fail to schedule $\Gamma_i \in \Gamma(\mathsf{HI})$ in HI mode if $R^*_i(\mathsf{HI}) > D_i$ and $\Gamma_j \in \Gamma(\mathsf{LO})$ in LO mode if $R^*_j(\mathsf{LO}) > D_j$. ∎

D. How to Determine $J^o_{i,k}$

While Algorithm 1 with the above scheduling policy provides a simple, but efficient interface for utilizing distributed resources, the protocol itself cannot provide the end-to-end delay guarantee. The guarantee can be achieved by carefully designing $J^o_{i,k}$, which needs the following requirements.

- *R1:* For $\Gamma_i \in \Gamma(\mathsf{HI})$, $J^o_{i,k}$ should be sufficiently small such that even if the inequality of line 1 in Algorithm 1 (i.e., $J_{i,k} > J^o_{i,k}$) holds, the corresponding job never misses its deadline in any case to satisfy the goal G1.
- *R2:* $J^o_{i,k}$ should satisfy the goal G2—maximizing the number of jobs in LO flows that meet end-to-end deadlines.

For R1, we need to determine $J^o_{i,k}$ so as to meet the deadline of a job of a HI flow in the presence of a mode change. At the same time, for R2, we need to minimize the number of mode changes. To this end, we now explain a policy for $J^o_{i,k}$ to capture the last minute to delay a mode change, called the Lazy policy. The Lazy policy for Γ_i on $s_{i,k}$ assumes the situation where the job of Γ_i of interest executes $s_{i,k}$ in LO mode but all the remaining stages in HI mode, which yields the minimum response time for the job to execute from the current stage to the last without changing the mode of $s_{i,k}$ to HI. If the worst-case response time for this situation (associated with a target schedulability test) exceeds the deadline of the job, we cannot guarantee the timely completion of the job on the last stage without triggering a mode change on $s_{i,k}$; in this case, we inevitably change the mode of $s_{i,k}$ from LO to HI. This yields the following criterion for the lazy policy:

$$J^o_{i,k} = D_i - R^*_{i,k}(\mathsf{LO}) - \sum_{k+1 \leq m \leq n_i} R^*_{i,m}(\mathsf{HI}). \tag{5}$$

The following theorem presents that the condition for the lazy policy is a feasibility condition for $J^o_{i,k}$.

Theorem 1: Given a distributed system $\mathbf{\Gamma}$ that satisfies the necessary condition of (4), it is feasible to schedule $\mathbf{\Gamma}$ correctly if $\forall \Gamma_i \in \mathbf{\Gamma}, \forall s_{i,k} \in S_i$

$$J^o_{i,k} \leq \begin{cases} \text{RHS of (5)}, & \text{if } \Gamma_i \in \Gamma(\mathsf{HI}) \\ \sum_{1 \leq m < k} R^*_{i,m}(\mathsf{LO}), & \text{otherwise.} \end{cases} \tag{6}$$

Note that *RHS* stands for the right-hand side of an equation. To prove Theorem 1, we present an auxiliary lemma for the property that for all HI flows $\Gamma_i, \Gamma_j \in \Gamma(\mathsf{HI})$, the analysis of $R^*_{i,k}(\mathsf{LO})$ remains true even if $\tau_{j,k'}$ triggers a mode change at stage $s_{i,k} = s_{j,k'}$.

Lemma 2: The response time $R_{i,k}$ of a HI mode step $\tau_{i,k}$ does not exceed $R^*_{i,k}(\mathsf{LO})$ even though it experiences a mode change that another HI flow Γ_j triggers on $s_{i,k}$.

Proof: Since Γ_i and Γ_j share the same stage $s_{i,k}$, there exists k' such that $s_{i,k} = s_{j,k'}$. By definition, Γ_j exhibited HI behavior on $s_{j,k'}$ by exceeding $J^o_{j,k'}$, i.e., $J_{j,k'} > J^o_{j,k'}$, and this might yield additional interference on Γ_i. We note that this is the only source of additional interference, since other possible sources like $C_{j,k}$ and T_j remain fixed.

As shown in (3), $R^*_{i,k}(\mathsf{LO})$ is calculated with the maximum possible jitter of higher priority flows Γ_j, $J^*_{j,k'}$, which is equal to $\sum_{1 \leq l < k'} R^*_{j,l}(\mathsf{LO})$ according to (2). Since each HI flow Γ_j finishes earlier than $R^*_{j,m}(\mathsf{LO})$ on any HI mode stage $s_{j,m}$, $R_{j,m}(\mathsf{LO})$ can never exceed $R^*_{j,m}(\mathsf{LO})$. Therefore, $\sum_{1 \leq l < k'} R_{j,m}(\mathsf{LO}) \leq \sum_{1 \leq m < k'} R^*_{j,m}(\mathsf{LO})$. Then, it is valid that $R_{i,k} \leq R^*_{i,k}(\mathsf{LO})$ since Γ_j arrives on $s_{j,k'}$ within $J^*_{j,k'}$ time units after the release on $s_{j,1}$.

If Γ_j is a LO flow violating $J^o_{j,k'}$, Γ_j has no effect on $R_{i,k}$ since it is dropped in this case. If Γ_j is a LO flow exhibiting LO behavior, it has no negative impact on $R^*_{i,k}(\mathsf{LO})$ since it arrives on $s_{j,k'}$ $(=s_{i,k})$ within $J^o_{j,k'}$ time units after the release on $s_{j,1}$, which is smaller than $\sum_{1 \leq m < k'} R^*_{j,m}(\mathsf{LO})$. ∎

More specifically, though $J^o_{j,k'}$ property is violated at $s_{i,k}$ triggering the mode change to HI mode, it does not affect the worst-case analysis, $R^*_{i,k}(\mathsf{LO})$.

Proof of Theorem 1: We prove the theorem by contradiction. Suppose even if $J^o_{i,k}$ satisfies Theorem 1 and the mode change framework is working properly, Γ_i missed its deadline, i.e.,

$$D_i < R_i. \tag{7}$$

We consider two cases depending on the criticality level of Γ_i: HI and LO.

For the first case where $\Gamma_i \in \Gamma(\mathsf{HI})$, we further consider two subcases depending on whether or not τ_{i,n_i} violates

J^o_{i,n_i} at the last stage s_{i,n_i}. Suppose it does not violate, i.e., $J_{i,n_i} \le J^o_{i,n_i}$. Lemma 2 states that the actual response time R_{i,n_i} does not exceed $R^*_{i,n_i}(\mathsf{LO})$ even though Γ_i experiences a mode change triggered by another HI flow. Then, it follows from the definition (6) that:

$$R_i = J_{i,n_i} + R_{i,n_i} \le J^o_{i,n_i} + R^*_{i,n_i}(\mathsf{LO}) \le D_i.$$

This contradicts (7).

Suppose Γ_i violates J^o_{i,n_i} (i.e., $J_{i,n_i} > J^o_{i,n_i}$) and triggers a mode change at s_{i,n_i}. Then, let $s_{i,k}$ be the first stage such that Γ_i continues to trigger mode changes on a series of stages $[s_{i,k}, \ldots, s_{i,n_i}]$. It is sufficient to consider these consecutive HI stages, since for LO mode stage $s_{i,k-1}$, $\sum_{1\le m<k-1} R_{i,m}(\mathsf{LO})$ is correctly upper-bounded by $J^o_{i,k-1}$ and it therefore will not affect the correctness. In the case where $k = 1$, Γ_i triggers mode changes from a source node, and it is clear that

$$R_i = J_{i,1} + \sum_{1\le m<n_i} R_{i,m}(\mathsf{HI}) + R_{i,n_i}(\mathsf{HI}) = 0 + R_i(\mathsf{HI}) \le D_i.$$

The last inequality holds from the necessary condition, (4), which contradicts (7).

In the case where $2 \le k \le n_i$, it follows from Lemma 2 and (6) that:

$$\begin{aligned} R_i &= J_{i,k} + \sum_{k\le m\le n_i} R_{i,m}(\mathsf{HI}) \\ &= J_{i,k-1} + R_{i,k-1} + \sum_{k\le m\le n_i} R_{i,m}(\mathsf{HI}) \\ &\le J^o_{i,k-1} + R^*_{i,k-1}(\mathsf{LO}) + \sum_{k\le m\le n_i} R^*_{i,m}(\mathsf{HI}) \\ &\le D_i \end{aligned}$$

which also contradicts (7).

For the second case where $\Gamma_i \in \Gamma(\mathsf{LO})$, we further consider two subcases in terms of the criticality level of a higher-priority flow Γ_j: HI and LO. Here, since the T and C terms are assumed to be fixed, we only consider the J term. With k' that satisfies $s_{i,k} = s_{j,k'}$:

1) consider Γ_j is a HI flow. If $J_{j,k'} > J^o_{j,k'}$, Γ_j triggers a mode change and the system does not guarantee the schedulability of Γ_i. Otherwise, we have $J_{j,k'} \le J^o_{j,k'}$. Since $J^o_{j,k'} \le J^*_{j,k'}$, the maximum possible interference of Γ_j on Γ_i is already captured in the analysis of $R^*_i(\mathsf{LO})$. It follows that $R_i \le R_i(\mathsf{LO}) \le D_i$, contradicting (7);
2) consider Γ_j is a LO flow. Let us assume Γ_j imposes a larger amount of interference on Γ_i at $s_{i,k}$ than the one calculated in the analysis of $R^*_i(\mathsf{LO})$. Since T_j and C_j are fixed, this assumption cannot be valid when $J_{j,k'} \le J^o_{j,k'}$. Since Γ_j is dropped whenever $J_{j,k'} > J^o_{j,k'}$, the above assumption cannot be valid. ∎

We now prove the optimality of Theorem 1 subject to given schedulability analysis.

Lemma 3: If a scheduler sets $J^o_{i,k}$ of $\Gamma_i \in \Gamma(\mathsf{HI})$ to a value larger than the RHS of (5), there exists a scenario that results in a job deadline miss for $\Gamma_i \in \Gamma(\mathsf{HI})$ subject to the target schedulability analysis.

Proof: Let RHS denote the RHS of (5). Suppose that a scheduler sets $J^o_{i,k}$ to $\mathtt{RHS}+\epsilon$. If $\mathtt{RHS} < J_{i,k} \le \mathtt{RHS}+\epsilon$ holds, JMC with the scheduler does not trigger a mode change for Γ_i on $s_{i,k}$. If $R_{i,k}$ on LO mode $s_{i,k}$ equals to $R^*_{i,k}(\mathsf{LO})$ and $R_{i,m}$ on HI mode $s_{i,m}$ equals to $R^*_{i,m}(\mathsf{HI})$ for every $k+1 \le m \le n_i$, a job of Γ_i misses its deadline. ∎

VI. Development of Efficient Jitter-Threshold Assignment Policies

In this section, we develop two jitter-threshold (i.e., $J^o_{i,k}$) assignment policies: 1) the Lazy and 2) Proactive policies. While both satisfy R1, the former and the latter are favorable to achieving R2 when the target schedulability test is pessimistic and tight, respectively. We prove that the two policies achieve R1 (without any condition) and R2 under some conditions.

A. Lazy Policy

One of the simplest ways to assign the threshold of a jitter for $\tau_{i,k}$ is to assign the largest possible value that does not compromise R1. This policy is called the *lazy* policy, as we explained in (5). Then, it is trivial that the lazy policy satisfies R1.

Lemma 4: If we apply JMC with the lazy policy to $\mathbf{\Gamma}$, then no job of $\Gamma_i \in \Gamma(\mathsf{HI})$ misses its deadline as long as $\mathbf{\Gamma}$ is feasible by Theorem 1. This is equivalent to satisfying R1.

Proof: The lemma holds by Theorem 1. ∎

While it seems that the lazy policy considers R1 only, the policy compensates the pessimism of the target schedulability analysis, by procrastinating mode changes as much as possible. Therefore, the Lazy policy achieves R2 if the actual response time of Γ_i on $s_{i,k}$ (i.e., $R_{i,k} = J_{i,k+1} - J_{i,k}$) is sufficiently smaller than the response time calculated by the target schedulability analysis [i.e., $R^*_{i,k}(\mathsf{LO})$], recorded by the following lemma.

Lemma 5: If the following inequality holds for every pair of $\Gamma_i \in \mathbf{\Gamma}$ and $1 \le k \le n_i - 1$, JMC with the Lazy policy for $\mathbf{\Gamma}$ achieves R2:

$$J_{i,k+1} - J_{i,k} \le R^*_{i,k}(\mathsf{LO}) + \left(R^*_{i,k+1}(\mathsf{LO}) - R^*_{i,k+1}(\mathsf{HI})\right). \quad (8)$$

Proof: Case I ($D_i - R^*_{i,1}(LO) - \sum_{m=2}^{n_i} R^*_{i,m}(HI) >= J_{i,1} = 0$): In this case, the Lazy policy does not trigger a mode change for Γ_i at the first stage of Γ_i (i.e., $s_{i,1}$). In the next stage $s_{i,2}$, the Lazy policy compares $D_i - R^*_{i,2}(\mathsf{LO}) - \sum_{m=3}^{n_i} R^*_{i,m}(\mathsf{HI})$ with $J_{i,2}$. Compared to the corresponding values to $s_{i,1}$, the former increases by $R^*_{i,1}(\mathsf{LO}) - \left(R^*_{i,2}(\mathsf{LO}) - R^*_{i,2}(\mathsf{HI})\right)$, and the latter increases by $J_{i,2} - J_{i,1}$. By applying (8), we know that the former is still larger than the latter, and therefore, the Lazy policy does not trigger a mode change for Γ_i at $s_{i,2}$. The relationship between $s_{i,1}$ and $s_{i,2}$ holds for $s_{i,k}$ and $s_{i,k+1}$ for all $2 \le k \le n_i - 1$, yielding no mode change at the rest of the stages. Since no mode change occurs, no other policy is better than the Lazy policy in meeting R2, which proves this case.

Case II ($D_i - R^*_{i,1}(LO) - \sum_{m=2}^{n_i} R^*_{i,m}(HI) < J_{i,1} = 0$): In this case, every policy including the Lazy policy triggers a mode change for Γ_i at its first stage $s_{i,1}$. Then, the problem of determining a mode change for the following stages is the

same as the original problem with replacing D_i with $D_i - (J_{i,2} - J_{i,1})$.

By the two cases, the lemma holds. ∎

B. Proactive Policy

While the Lazy policy postpones mode changes of stages that Γ_i passes as much as possible, we develop another policy that considers the number of LO-jobs that are affected upon the mode change triggered by Γ_i. This policy, namely Proactive, may trigger mode switch proactively if the cost of mode switch is low enough on the current stage.

To this end, we first define the *cost* of the mode switch. When the stage is in HI mode, the jobs of LO flows may be dropped. We denote the upper bound of the number of LO jobs that are affected by the mode change triggered by Γ_i on $s_{i,k}$. Then, $\delta_{i,k}$ can be computed as follows:

$$\delta_{i,k} = \sum_{\Gamma_j \in \Gamma(\mathsf{LO}) \text{ and } s_{i,k} \in S_j} \left\lceil \frac{R^*_{i,k}(\mathsf{HI})}{T_j} \right\rceil. \tag{9}$$

With $\delta_{i,k}$, the Proactive policy repeats to assign the HI mode stages with the lowest $\delta_{i,k}$ first until the end-to-end response time (calculated by the target schedulability analysis) meets its deadline. This decision takes place at every stage, and a mode change occurs if the current stage $s_{i,k}$ is assigned to be HI stage. Formally, this goal can be stated as the following:

$$\text{A mode change occurs in } s_{i,k}, \text{ if } s_{i,k} \in \hat{S}_i(m)$$

where m is the minimum number of HI assigned stages

$$\text{s.t. } J_{i,k} + \sum_{s_{i,r} \in \hat{S}_i(m)} R^*_{i,r}(\mathsf{HI}) + \sum_{s_{i,t} \in S_i \setminus \hat{S}_i(m)} R^*_{i,t}(\mathsf{LO}) \le D_i. \tag{10}$$

Here, $\hat{S}_i(m)$ is the top-m elements of remaining stages $[s_{i,k}, \ldots, s_{i,n_i}]$, sorted by $\delta_{i,k}$ in nondecreasing order, where $0 \le m \le n_i - k$. By solving the minimum m in (10), we can select the low-cost mode switches in a greedy manner until Γ_i meets its end-to-end deadline. Then, we make a decision for the mode switch checking whether the current stage $s_{i,k}$ is assigned to be HI mode. We can express this policy using the $J^o_{i,k}$ term as follows:

$$J^o_{i,k} = D_i - \sum_{\delta_{i,r} < \delta_{i,k}} R^*_{i,r}(\mathsf{HI}) - \sum_{\delta_{i,t} \ge \delta_{i,k}} R^*_{i,t}(\mathsf{LO}) \tag{11}$$

where $k \le r, t \le n_i$. In other words, we can get $J^o_{i,k}$ by subtracting $R^*_{i,r}$ of remaining stages from D_i, assuming HI mode on lower-cost stages and LO mode on higher-cost stages than the current stage.

The following lemma proves the equivalence of mode change decisions for Γ_i on $s_{i,k}$ based on (10) and (11).

Lemma 6: Equation (10) decides to trigger a mode change for Γ_i on $s_{i,k}$ if and only if $J_{i,k}$ is larger than $J^o_{i,k}$ in (11).

Proof: Denote l as the rank of $s_{i,k}$ among remaining stages, sorted by $\delta_{i,k}$ in nondecreasing order. Also, let the solution of (10) be $\hat{m}$. Then, (10) makes a mode change for Γ_i on $s_{i,k}$ if and only if $\hat{m} \ge l$. We can rewrite (10) as

$$J_{i,k} \le D_i - \sum_{s_{i,r} \in \hat{S}_i(m)} R^*_{i,r}(\mathsf{HI}) - \sum_{s_{i,t} \in S_i \setminus \hat{S}_i(m)} R^*_{i,t}(\mathsf{LO}) \tag{12}$$

and denote the right-hand side of (12) as $f(m)$. Then, $f(m)$ is an increasing function on m. Therefore, the solution $\hat{m}$ should satisfy $J_{i,k} \le f(\hat{m})$ and $J_{i,k} > f(\hat{m}-1)$. Then, the mode change condition from (10) becomes

$$\hat{m} \ge l \Leftrightarrow f(\hat{m}-1) \ge f(l-1) \Leftrightarrow J_{i,k} > f(l-1)$$
$$\hat{m} < l \Leftrightarrow \hat{m} \le l-1 \Leftrightarrow f(\hat{m}) \le f(l-1) \Leftrightarrow J_{i,k} \le f(l-1).$$

Since we can rewrite the right-hand side of (11) as $f(l-1)$, (10) and (11) take the same decision in any case. ∎

We next prove that the Proactive policy does not compromise R1.

Lemma 7: $J^o_{i,k}$ set by the Proactive policy is no larger than $J^o_{i,k}$ set by the Lazy policy, implying JMC with the Proactive policy satisfies R1.

Proof: Since we have the inequality $R^*_{i,k}(\mathsf{HI}) \le R^*_{i,k}(\mathsf{LO})$, the maximum value of (11) occurs when $\delta_{i,k}$ is the maximum among $\delta_{i,r}$, where $k \le r \le n_i$. In this case, (11) becomes

$$J^o_{i,k} = D_i - R^*_{i,k}(\mathsf{LO}) - \sum_{k+1 \le r \le n_i} R^*_{i,r}(\mathsf{HI}).$$

This is exactly $J^o_{i,k}$ set by the Lazy policy. ∎

Finally, we demonstrate that the Proactive policy is advantageous for achieving R2, by proving its achievement for R2 under some conditions.

Lemma 8: Suppose that the actual response time is always the same as the worst-case response time by the target schedulability analysis. In addition, suppose that the difference between $R^*_{i,k}(\mathsf{LO}) - R^*_{i,k}(\mathsf{HI})$ is uniform along the stages $s_{i,k}, \ldots, s_{i,n_i}$. Then, JMC with the Proactive policy for Γ_i minimizes the number of LO jobs that are affected by mode changes triggered by Γ_i, if $\delta_{i,k}$ in (9) yields the exact number of jobs that are affected by mode changes triggered by Γ_i on $s_{i,k}$.

Proof: When we have full knowledge of actual response times of next stages as $R^*_{i,k}(\mathsf{LO})$ and $R^*_{i,k}(\mathsf{HI})$, there exists the optimal mode assignment that minimizes $\sum_m \delta_{i,m}$ of HI mode stages, satisfying the end-to-end deadline. Suppose the situation where a job runs in LO mode at the rest of the stages from $s_{i,k}$. Then our end-to-end response time will be $J_{i,k} + \sum_{k \le m \le n_i} R^*_{i,m}(\mathsf{LO})$, which may not meet the deadline D_i.

If we decide to change the mode of stage $s_{i,m'}$, the response time is decreased by $R^*_{i,m'}(\mathsf{LO}) - R^*_{i,m'}(\mathsf{HI})$, with the cost $\delta_{i,m'}$. When the decreased amount of time by mode change is uniform among the stages, the number of stages assigned to HI mode is fixed. In this condition, the optimal policy minimizing $\sum_m \delta_{i,m}$ of HI stages is the same as (10): selecting the stages with lower $\delta_{i,m}$ first.

In addition, if $\delta_{i,k}$ estimation is exact, our Proactive policy will also minimize the overall number of jobs that experience HI mode stage along the path S_i due to Γ_i, since it finds the optimal solution that minimizes $\sum_m \delta_{i,m}$. ∎

Although Proactive in general cases produces suboptimal solutions to minimize the number of LO jobs that are affected by mode changes triggered by Γ_i, it approximates the optimal solution and produces better decisions than Lazy when the analysis is accurate. Also, R2 is highly related to minimize the number, and therefore, the Proactive policy is favorable in achieving R2 when the analysis is accurate.

In summary, while the Lazy and Proactive policies achieve R1 (proven by Lemmas 4 and 7), the Lazy and Proactive policies have advantages in achieving R2, when the target schedulability analysis is pessimistic and tight (proven by Lemmas 5 and 8 with some conditions), respectively.

VII. Evaluation

We now demonstrate the effectiveness of our proposed scheduling framework and jitter-threshold assignment policies in terms of achieving G1 and G2.

Simulation Setup: We conducted various simulations based on a distributed system that consists of 16 computation nodes connected with each other by communication links, in a 4×4 grid topology. For each simulation, we generated a random flow set as follows. The number of flows in a flow set (denoted as $N_{\mathbf{\Gamma}}$) was chosen from 5 to 50 with a step size of 5. Note that our choice for $N_{\mathbf{\Gamma}}$ aims at representing the system load as in many network scheduling studies [13], [14], since the system utilization does not capture the load in distributed systems effectively. For each flow Γ_i, the first stage $s_{i,1}$ was randomly selected from the computation nodes, and the last stage s_{i,n_i} was randomly chosen among the computation nodes reachable from $s_{i,1}$ within the link-hop distance of 4 (i.e., the link-hop distance is 0, 1, 2, 3, or 4). Note that the number of stages including computing nodes and network links (n_i) was determined between 1 and 9, according to the hop distance between the first and last stages. The period T_i was randomly chosen between 10 and 200 (with $D_i = T_i$) according to log-uniform distribution, and its execution/transmission time $C_{i,k}$ was randomly determined between 0.15 * T_i/n_i and 0.3 * T_i/n_i, also according to log-uniform distribution. The criticality level L_i was set to HI with the probability parameter (denoted as P_{HI}), where P_{HI} was set from 0.1 to 0.9 with a step size of 0.1. We assume that a communication link can transmit up to the size of MTU (i.e., up to one packet) per unit-time. Since a communication link cannot preempt each packet transmission, each link has the blocking time $B_{i,k}$ of 1 (i.e., maximum time unit to transmit the nonpreemptible packet).

A. Schedulability Ratio

In this section, we compared the *schedulability ratio* of JMC and *criticality-aware scheduling* including CA-DM, which is defined as the ratio of *schedulable* flow sets to the total number of generated flow sets. Here, we counted a flow set as schedulable under JMC, if the response time analysis guarantees the JMC-schedulability (defined in Section IV) of the flow set. On the other hand, we counted a flow set as schedulable under criticality-aware scheduling, if the response time analysis guarantees that the end-to-end response time of every flow in the flow set is no larger than its relative deadline. We evaluated the schedulability ratio with varying the number of flows $N_{\mathbf{\Gamma}}$ and the probability parameter $\boldsymbol{P}_{\mathsf{HI}}$.

To evaluate JMC compared to other scheduling algorithms, we measured the performance of JMC(X) and CA-X, where JMC(X) and CA-X denote JMC and criticality-aware scheduling with the priority assignment X, respectively.[5] Note that in CA-X, flows are assigned their priority according to: 1) criticality (i.e., HI flows are always higher than LO flows) and 2) the criterion used by X. Note that, to the best of our knowledge, JMC is the first study to consider the JMC scheduling; thereby, it is natural to compare with and without the jitter-based scheduling [i.e., JMC(X) and CA-X, respectively]. With them, we applied various flow-level fixed priority assignment algorithms as follows.

1) *RD:* RanDom; random priority assignment.
2) *DM:* Deadline monotonic; the smaller the deadline (D_i), the higher the priority.
3) *SLM:* Static laxity monotonic; the smaller the static laxity ($D_i - C_i$), the higher the priority.
4) *PSLM:* Per-stage static laxity monotonic; the smaller the per-stage static laxity ($[(D_i - C_i)/n_i]$), the higher the priority.

Note that we used RD as the minimum baseline. Except RD, all algorithms are intuitively expected to be effective in distributed systems, since they effectively capture the urgency of each end-to-end flow. For instance, PSLM captures the time slot allowed to be interfered on each stage (i.e., $[(D_i - C_i)/n_i]$), that directly affects the schedulability of each flow.

1) Varying $N_{\mathbf{\Gamma}}$: Fig. 5(a) depicts the schedulability of each algorithm with varying $N_{\mathbf{\Gamma}}$ from 5 to 50. As $N_{\mathbf{\Gamma}}$ increases, the average utilization of each stage increases, which affects the schedulability. For each $N_{\mathbf{\Gamma}}$, we generated 1000 flow sets and determined the schedulability of those flow sets by applying each scheduling algorithm. In the figure, we observe that JMC(X) outperforms CA-X for every X, and the improvement of JMC(X) over CA-X is significant for every X except RD. This is expected since CA-X requires LO flows to meet their end-to-end deadline even in the situation of receiving interference from all HI flows, whereas JMC(X) only requires LO flows to do so with a smaller set of higher-priority HI and LO flows. In particular, JMC(PSLM) outperforms other algorithms; this is because PSLM is much effective to capture the urgency of each flow, with considering not only the static laxity ($D_i - C_i$) but also the number of stages (n_i). Meanwhile, CA-PSLM is not advantageous as much as JMC(PSLM); it only shows slightly higher performance than other CA-X algorithms. This is because, even in CA-PSLM, LO flows still suffer from interference by all HI flows. The result not only shows that JMC(X) outperforms CA-X regardless of X, but also establishes the expectation that JMC can benefit from other priority assignment algorithms working well in distributed systems.

2) Varying P_{HI}: Fig. 5(b) shows the schedulability with varying P_{HI} to control the ratio of HI and LO flows in each

[5]Note that comparison between jitter-threshold assignment policies is excluded for this experiment since jitter-threshold assignment policies do not affect schedulability performance. We will present their comparison in terms of other metrics in Sections VII-B and VII-C.

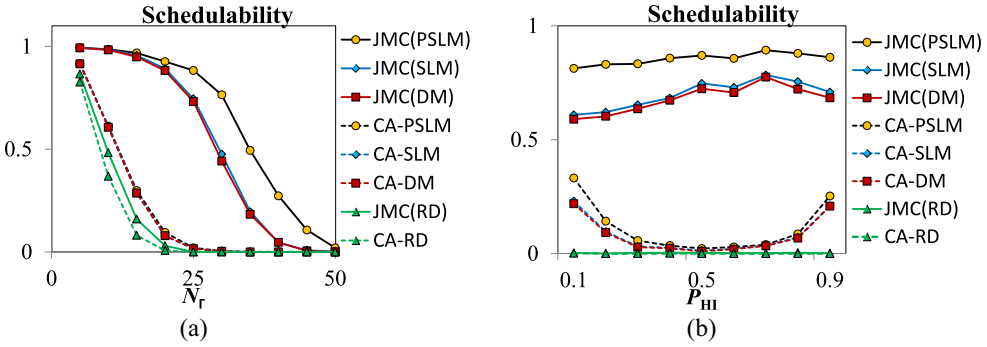


Fig. 5. Schedulability ratio. Varying (a) N_Γ and (b) P_{HI}.

flow set. P_{HI} was set from 0.1 to 0.9 with a step size of 0.1. For each P_{HI}, we generated 1000 flow sets with $N_\Gamma = 25$. We observe that JMC(X) outperforms CA-X regardless of X, except RD. In particular, the gap becomes larger when P_{HI} is around 0.5. In CA-X, the schedulability of LO flows is directly affected by the number of HI flows because a LO flow is interfered by all HI flows on each stage. Therefore, as P_{HI} increases, the schedulability decreases. However, from some point (i.e., $P_{\mathsf{HI}} = 0.5$), the schedulability increases again. This is because flow sets have a smaller number of LO flows that are contending with HI flows. As to JMC(X), when $P_{\mathsf{HI}} \leq 0.5$, the schedulability of the flow set constantly increases since JMC(X) can effectively favor HI flows to be scheduled to meet their deadlines. On the other hand, when $P_{\mathsf{HI}} > 0.5$, the schedulability of the flow set decreases since the increased number of HI flows makes them difficult to be schedulable together to meet their guarantees (G1). Similar to Fig. 5(a), we observe that JMC(PSLM) outperforms other algorithms; this is because PSLM makes the favorable priority assignment in distributed systems. With the advantage of PSLM, CA-PSLM results in slightly better performance than CA-DM and CA-SLM, in particular when P_{HI} is 0.1 or 0.9. However, it still shows low schedulability ratio compared to JMC(X).

B. Number of Schedulable LO Jobs

In order to evaluate how well each scheme accommodates LO flows, we measured *the number of schedulable LO jobs*, which is defined as the number of LO jobs that meet their end-to-end deadlines. We ran simulations for $\min\{LCM_\Gamma, 10^6\}$ time units, where LCM_Γ is the least common multiplier of periods of all flows in a flow set Γ. For ease of presentation, we present the result with the DM priority assignment algorithm as a representative; we observed that the results show similar trends regardless of the priority assignment algorithm.

1) Varying N_Γ: Fig. 6(a) shows the number of schedulable LO jobs with varying the number of flows N_Γ from 5 to 50. Note that the result of each approach is normalized to that of CA-DM. For each N_Γ, 20 flow sets were selected from randomly generated flow sets with $P_{\mathsf{HI}} = 0.5$, where both CA-DM and JMC(DM) guarantee all HI flows to meet the deadlines. The figure shows that JMC(DM) allows more LO jobs (up to 13.2% more than CA-DM) to complete before their deadlines. Under CA-DM, each LO flow is interfered by all HI flows, while,

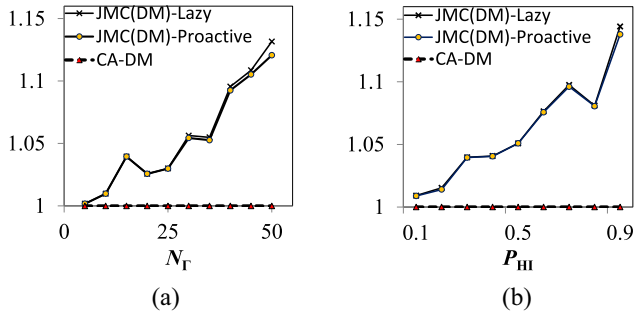


Fig. 6. Normalized number of schedulable LO jobs. Varying (a) N_Γ and (b) P_{HI}.

under JMC(DM), it is interfered by a subset of HI flows having a shorter deadline than the LO flow when it passes LO mode stages. Therefore, under JMC(DM), LO flows can have a shorter response time $R_{i,k}$, resulting in a higher chance to meet the end-to-end deadline, compared to CA-DM. As N_Γ increases, under CA-DM, LO flows are interfered by more HI flows, while they are interfered by a much smaller number of HI flows under JMC(DM). Note that, under JMC(DM), LO flows can be penalized when it passes HI mode stages. However, we observe that a mode change from LO to HI rarely happens at runtime because each HI flow rarely violates its optimistic release jitters, $J^o_{i,k}$. Therefore, JMC(DM) has more schedulable LO flows than CA-DM.

2) Varying P_{HI}: Fig. 6(b) shows the number of schedulable LO jobs with varying P_{HI} from 0.1 to 0.9. The result of each approach is also normalized to that of CA-DM. For each P_{HI}, 20 flow sets were selected from randomly generated flow sets with $N_\Gamma = 30$, where both CA-DM and JMC(DM) guarantee all HI flows to meet the deadlines. The figure shows that JMC(DM) allows more LO jobs (up to 14.4% more than CA-DM) to meet their end-to-end deadlines. This is expected because of the similar reason we observed in Fig. 6(a). As P_{HI} increases, more HI flows join the system. Under CA-DM, LO flows suffer from interference by all HI flows, while they are only interfered by a smaller number of HI flows under JMC(DM).

In Fig. 6(a), we observe that JMC(DM)-Lazy has more schedulable LO flows than JMC(DM)-Proactive when $N_\Gamma = 50$. This is because the flow sets contain a high degree of pessimism in the analysis accumulated along the stages. JMC(DM)-Proactive

decides a mode change under the assumption that all remaining stages exhibit the worst-case behavior in terms of response time; however, the worst case rarely happens. In contrast, JMC(DM)-Lazy postpones mode change as late as possible, based on the expectation that $R_{i,k}$ will be smaller than $R^*_{i,k}$ so that the mode change will not be necessary. With this principle, JMC(DM)-Lazy makes the smaller number of mode changes than JMC(DM)-Proactive, in particular when the difference between $R^*_{i,k}$ and $R_{i,k}$ is large [e.g., when $N_{\mathbf{\Gamma}} = 50$ in Fig. 6(a)]. We will further investigate the characteristics of two algorithms in the following section.

C. Comparison on Jitter-Threshold Assignment Policies

We observe that the Lazy policy shows better performance than the Proactive policy in some cases. However, since the Lazy policy does not consider the penalty of a mode change specific to each stage, it inherently risks performance degradation, in particular, depending on the degree of pessimism in the analysis.

To examine the performance difference between the two policies, we simulated a situation where a different number of LO flows execute on different stages. We consider a distributed system which consists of two computation nodes connected by a communication link. In the system, a HI flow Γ_i went through three stages $s_{i,1}$, $s_{i,2}$, and $s_{i,3}$, where $s_{i,1}$ and $s_{i,3}$ were computation nodes, and $s_{i,2}$ was a communication link. On the stages $s_{i,1}$ and $s_{i,3}$, five and seven LO flows were executed, respectively, with the higher priority than Γ_i; thereby, the penalty of mode change on $s_{i,3}$ is greater than on $s_{i,1}$. To investigate the effectiveness of each jitter-threshold assignment policy according to the analysis pessimism, we consider two cases: harmonic and nonharmonic period flow sets. Nonharmonic period flow sets typically attribute more pessimism, and we also experimentally confirmed that the average difference between actual response time R_i and worst-case response time estimate R^*_i was 20% greater in the nonharmonic sets than in the harmonic sets. Note that we used the DM priority assignment as a representative, since simulations with other priority assignment algorithms also bring an identical implication.

Fig. 7 plots the number of schedulable LO jobs with varying D_i of the HI flow Γ_i. Note that the result is normalized to the total number of LO jobs in the flow set. The x-axis represents the *flow density* of Γ_i, which is calculated by C_i/D_i where C_i is the total execution/transmission time over all stages. As the deadline gets tighter (i.e., decrease in D_i), the flow density increases and it becomes more likely to trigger mode changes.

As shown in Fig. 7(a), the Proactive policy selects the mode changing stage that results in fewer LO job drops (i.e., $s_{i,1}$), when the mode change has to be taken place (i.e., when the density becomes higher). Therefore, the Proactive policy makes a robust decision against the D_i value, making relatively good performance regardless of flow density. On the other hand, the Lazy policy that is oblivious to such information, delays the decision until the last minute (i.e., mode change on $s_{i,3}$) leading to more LO job drops. Accordingly, the result shows that the Proactive policy is able to dominate the Lazy policy when the flow density of Γ_i is larger. Meanwhile, the performance of the

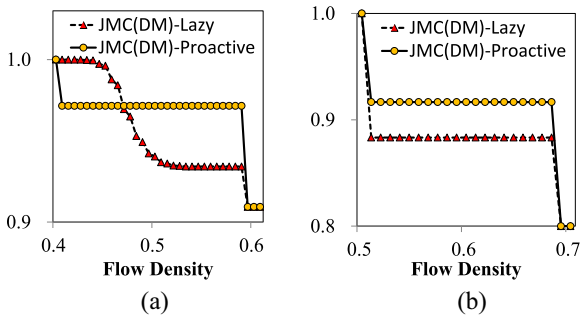


Fig. 7. Normalized number of schedulable LO jobs with varying the flow density. (a) Nonharmonic set. (b) Harmonic set.

Lazy policy gradually improves as the flow density decreases (i.e., increase in D_i). This improvement mainly comes from a large pessimism, leading to a larger difference between actual response times and worst-case response time estimates. Such a difference makes it advantageous to delay mode change decision-making, as stated in Lemma 5. The timing gain leaves a room for the Lazy policy not to change its mode even at $s_{i,3}$; in addition, as the flow density decreases, the Lazy policy has more rooms (i.e., more time slots until the deadline) to keep all stages as LO mode. However, when such an effect is reduced in the harmonic set as shown in Fig. 7(b), the Lazy policy is more penalized in making decisions. In fact, the Lazy policy does not do any better than the Proactive policy. As stated in Lemma 8, as the actual response times get closer to a WCRT estimate, the Proactive policy becomes more effective.

In summary, the performance of each policy depends on tightness of the WCRT analysis. The Lazy policy performs relatively well in most cases which have a highly pessimistic WCRT analysis, e.g., when $N_{\mathbf{\Gamma}} = 50$ in Fig. 6(a). However, as the analysis becomes tighter, the significance of the Proactive policy may become greater. The Lazy and Proactive policies can be selectively applied according to the system setup; note that the pessimism in the analysis depends on the system setup (see Section III).

VIII. Discussion

So far, JMC has focused on MC scheduling based on the jitter parameter, while most MC scheduling have focused on the execution time parameter. Yet, it is possible to extend JMC with other parameters; in particular, JMC can be extended to consider the WCET parameter (the jitter is orthogonal to WCET). This can further alleviate the pessimism of the response time analysis, since it can use more optimistic WCET value for LO mode [i.e., $C_{i,k}(\text{LO})$]. Note that JMC always uses a single $C_{i,k}$ parameter estimated by HI mode behavior.

We first assume that each HI flow Γ_i has both $C_{i,k}(\text{HI})$ and $C_{i,k}(\text{LO})$ on each stage $s_{i,k}$. For ease of presentation, we define two properties as follows.

1) A job of Γ_i has a $C_{i,k}(\text{LO})$ property if it executes no more than $C_{i,k}(\text{LO})$ on stage $s_{i,k}$.
2) A job of Γ_i has a $J^o_{i,k}$ property if it arrives at stage $s_{i,k}$ no later than its release time plus $J^o_{i,k}$.

We then define job behavior associated with the properties. A job of a flow Γ_i is said to *exhibit LO behavior* if it satisfies both $C_{i,k}(\mathsf{LO})$ and $J^o_{i,k}$ properties. The job is said to *exhibit HI behavior* otherwise, i.e., if it violates either $C_{i,k}(\mathsf{LO})$ or $J^o_{i,k}$ property. With the new definition of job behavior, we can directly extend JMC to trigger a stage-level mode change based on not only jitter estimates but also WCET ones.

In order to validate the correctness of JMC under the new definition of job behavior, we also need to extend response time analysis to consider both jitter and WCET estimates. In particular, we focus on how to extend $R^*_{i,k}(\mathsf{HI})$ and $R^*_{i,k}(\mathsf{LO})$, which are core elements in response time analysis. First, recall that $R^*_{i,k}(\mathsf{HI})$ is the WCRT of Γ_i when $s_{i,k}$ is in HI mode. We can redefine $R^*_{i,k}(\mathsf{HI})$ by replacing $C_{i,k}$ and $hp(i,k)$ in (1) with $C_i(\mathsf{HI})$ and $hp(i,k,\mathsf{HI})$, respectively. Second, $R^*_{i,k}(\mathsf{LO})$ is the WCRT of Γ_i when $s_{i,k}$ is in LO mode. For $R^*_{i,k}(\mathsf{LO})$, the worst-case happens when a job of Γ_i violates the $C_{i,k}(\mathsf{LO})$ property in the middle of execution. Thus, $R^*_{i,k}(\mathsf{LO})$ should be redefined for HI flows. We can compute $R^*_{i,k}(\mathsf{LO})$ similarly with the methods in [5], and it can be presented as the following fixed-point iteration:

$$R^{*\,(n+1)}_{i,k}(\mathsf{LO}) = C_{i,k}(\mathsf{HI}) + B_{i,k} + \sum_{\tau_{j,k'} \in \boldsymbol{hp}(i,k,\mathsf{HI})} \left\lceil \frac{R^{*\,(n)}_{i,k}(\mathsf{LO}) + J^*_{j,k'}(\mathsf{LO})}{T_j} \right\rceil C_{j,k'}(\mathsf{HI}) + \sum_{\tau_{m,k''} \in \boldsymbol{hp}(i,k,\mathsf{LO})} \left\lceil \frac{R^{\mathsf{LO}}_{i,k}(\mathsf{LO}) + J^*_{m,k''}(\mathsf{LO})}{T_m} \right\rceil C_{m,k''}(\mathsf{LO})$$

where $R^{\mathsf{LO}}_{i,k}$ is the worst-case response time computed with $C_{i,k}(\mathsf{LO})$; $R^{*\,(0)}_{i,k}(\mathsf{LO})$ is set to 0; and the iteration ends when $R^{*\,(n)}_{i,k}(\mathsf{LO}) = R^{*\,(n+1)}_{i,k}(\mathsf{LO})$ or $R^{*\,(n)}_{i,k}(\mathsf{LO}) > D_i$ (deemed unschedulable). Note that a more efficient analysis method can be found in [5].

In addition, we can directly use the Lazy and Proactive policies. We can plug the newly defined $R^*_{i,k}(\mathsf{HI})$ and $R^*_{i,k}(\mathsf{LO})$ above into (5) for Lazy and (11) for Proactive, respectively, without compromising the properties of the policies.

IX. Related Work

MC systems have been introduced to address the problem of low CPU utilization. Since Vestal's [4] seminal work, a large body of work has been proposed for MC systems (see [23] for an extensive survey). In fixed-priority scheduling, Vestal [4] developed response time analysis of MC task model. Later, Baruah *et al.* [5] considered the runtime behavior of tasks and introduced criticality mode at runtime to improve resource efficiency. The concept of MC is further extended with the period and the relative deadline [6]–[8]. Although early MC studies ignored the performance of LO tasks when the low-criticality assumption is no longer valid, the performance of LO tasks is still important in the practical point of view [6]. Instead of suspending all LO tasks in HI mode, recent MC studies proposed to provide the degraded service for all LO tasks [6]–[8] or selective-dropping of LO tasks [9], [10].

MC scheduling has been extended to distributed systems. For classical distributed systems, there exist many analysis methods for response time of end-to-end flows, including RTA-based approach (e.g., holistic schedulability analysis [16], [20] and offset-based analysis [18]), RTC [17], and compositional performance analysis [19]. Among them, RTA-based approaches have been extended with MC on network platforms such as control-area network (CAN) [12] and network-on-chip (NoC) [13], [14]; those existing MC network studies adopted the system-wide [12] or path-wide mode change [13], [14].

In summary, this paper can be differentiated from the studies listed above as follow. First, none of the previous work has investigated applying the notion of MC scheduling to measure and respond to the pessimism in the worst-case interference (response time) analysis, while focusing on other static parameters such as WCET, the period, and the relative deadline. Second, this paper proposes a stage-level mode change mechanism to minimize the penalty of mode change (in terms of penalizing LO flows) for distributed systems, while the existing MC network studies adopted the system-wide [12] or path-wide [13], [14] mode change mechanism.

X. Conclusion

In this paper, we have presented JMC, an efficient JMC scheduling framework for distributed systems, to achieve the goal of maximizing the performance of LO flows while guaranteeing the schedulability of HI flows. JMC uses jitters to efficiently measure the pessimism in the analysis at runtime and enables the change of criticality mode on a per-stage basis to minimize the effect of the mode change. We have presented an optimal feasibility condition (subject to given schedulability analysis) to assign jitter-thresholds and introduced two assignment policies, each achieving its own goals. Our simulation results have shown that JMC has higher schedulability while accommodating more LO flows, compared to the existing criticality-aware fixed priority assignment scheduling schemes. In the future, we would like to extend JMC toward WCET (that has been discussed in Section VIII) and other parameters such as the period. In addition, although JMC focuses on dual-criticality systems, the scheduling framework can be extended to more than two criticality levels, by applying per-level jitter threshold. We leave it as future work.

References

[1] R. Wilhelm *et al.*, "The worst-case execution-time problem—Overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, pp. 1–53, Apr. 2008.

[2] P. Ekberg and W. Yi, "Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Dec. 2017, pp. 139–146.

[3] A. Burns, "Scheduling hard real-time systems: A review," *Softw. Eng. J.*, vol. 6, no. 3, pp. 116–128, May 1991.

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

[5] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, 2011, pp. 34–43.

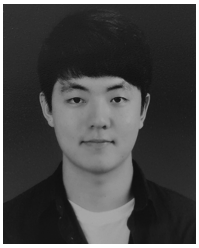
[6] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Proc. Workshop Mixed Criticality Syst. (WMC)*, 2013, pp. 1–6.

[7] O. Gettings, S. Quinton, and R. I. Davis, "Mixed criticality systems with weakly-hard constraints," in *Proc. Int. Conf. Real Time Netw. Syst. (RTNS)*, 2015, pp. 237–246.

[8] H. Su, P. Deng, D. Zhu, and Q. Zhu, "Fixed-priority dual-rate mixed-criticality systems: Schedulability analysis and performance optimization," in *Proc. IEEE Int. Conf. Embedded Real Time Comput. Syst. Appl. (RTCSA)*, Aug. 2016, pp. 59–68.
[9] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *Proc. Euromicro Conf. Real Time Syst. (ECRTS)*, Jul. 2015, pp. 13–24.
[10] J. Lee, H. S. Chwa, L. T. X. Phan, I. Shin, and I. Lee, "MC-ADAPT: Adaptive task dropping in mixed-criticality scheduling," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5S, pp. 1–21, Oct. 2017.
[11] *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, OMG document formal/11-06-02, Object Manag. Group, Needham, MA, USA, 2011.
[12] A. Burns and R. I. Davis, "Mixed criticality on controller area network," in *Proc. Euromicro Conf. Real Time Syst. (ECRTS)*, Jul. 2013, pp. 125–134.
[13] A. Burns, J. Harbin, and L. S. Indrusiak, "A wormhole NoC protocol for mixed criticality systems," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Dec. 2014, pp. 184–195.
[14] L. S. Indrusiak, J. Harbin, and A. Burns, "Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip," in *Proc. Euromicro Conf. Real Time Syst. (ECRTS)*, Jul. 2015, pp. 47–56.
[15] J. M. Rivas, J. J. Gutierrez, J. C. Palencia, and M. G. Harbour, "Schedulability analysis and optimization of heterogeneous EDF and FP distributed real-time systems," in *Proc. Euromicro Conf. Real Time Syst. (ECRTS)*, 2011, pp. 195–204.
[16] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, nos. 2–3, pp. 117–134, 1994.
[17] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 4, 2000, pp. 101–104.
[18] J. Mäki-Turja and M. Nolin, "Efficient implementation of tight response-times for tasks with offsets," *Real Time Syst.*, vol. 40, no. 1, pp. 77–116, Oct. 2008.
[19] R. Hofmann, L. Ahrendts, and R. Ernst, "CPA: Compositional performance analysis," in *Handbook of Hardware/Software Codesign*. Dordrecht, The Netherlands: Springer, 2017, pp. 721–751.
[20] J. C. Palencia and M. G. Harbour, "Offset-based response time analysis of distributed systems scheduled under EDF," in *Proc. Euromicro Conf. Real Time Syst. (ECRTS)*, Jul. 2003, pp. 3–12.
[21] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Softw. Eng. J.*, vol. 8, no. 5, pp. 284–292, Sep. 1993.
[22] R. Garibay-Martínez, G. Nelissen, L. L. Ferreira, P. Pedreiras, and L. M. Pinho, "Improved holistic analysis for fork–join distributed real-time tasks supported by the FTT-SE protocol," *IEEE Trans. Ind. Informat.*, vol. 12, no. 5, pp. 1865–1876, Oct. 2016.
[23] A. Burns and R. Davis. (2018). *Mixed Criticality Systems—A Review*. [Online]. Available: http://www-users.cs.york.ac.uk/burns/review.pdf

Kilho Lee received the B.S. degree in information and computer engineering from Ajou University, Suwon, South Korea, in 2010, and the M.S. and Ph.D. degrees in computer science from KAIST, Daejeon, South Korea, in 2012 and 2019, respectively.

He is a Post-Doctoral Researcher with the School of Computing, KAIST. His current research interests include system design and implementation for real-time embedded systems and cyber-physical systems.



Minsu Kim received the B.S. degree in computer science and engineering and the B.A. degree in political science and international relations from Seoul National University, Seoul, South Korea, in 2017. He is currently pursuing the M.S. degree in computer science with KAIST, Daejeon, South Korea.

Hayeon Kim received the B.S. degree in chemistry (with a minor in computer science) from KAIST, Daejeon, South Korea, in 2017, and the M.S. degree from the School of Computing, KAIST, in 2019.

Hoon Sung Chwa (GS'09–M'18) received the B.S., M.S., and Ph.D. degrees from KAIST, Daejeon, South Korea, in 2009, 2011, and 2016, respectively, all in computer science.

He is an Assistant Professor with the Department of Information and Communication Engineering, DGIST, Daegu, South Korea. He has been a Research Fellow with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA, until 2018. His current research interests include system design and analysis with timing guarantees and resource management in real-time embedded systems and cyber-physical systems.

Dr. Chwa was a recipient of the Best Paper Award from the 33rd IEEE Real-Time Systems Symposium in 2012 and the IEEE International Conference on Cyber-Physical Systems, Networks, and Applications in 2014.

Jaewoo Lee received the M.S. degree in computer science and engineering from Seoul National University, Seoul, South Korea, in 2008, and the Ph.D. degree in computer and information science from the University of Pennsylvania, Philadelphia, PA, USA, in 2017.

He is currently an Assistant Professor with Chung-Ang University, Seoul. His current research interests include cyber-physical systems, real-time embedded systems, and model-driven engineering.

Jinkyu Lee (GS'07–M'10) received the B.S., M.S., and Ph.D. degrees in computer science from KAIST, Daejeon, South Korea, in 2004, 2006, and 2011, respectively.

He joined Sungkyunkwan University, Suwon, South Korea, in 2014, where he is an Assistant Professor with the Department of Computer Science and Engineering. He has been a Research Fellow/Visiting Scholar with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA, until 2014. His current research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems and cyber-physical systems.

Dr. Lee was a recipient of the Best Student Paper Award of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium in 2011 and the Best Paper Award of the 33rd IEEE Real-Time Systems Symposium in 2012.

Insik Shin (M'11) received the B.S. degree in computer science from Korea University, Seoul, South Korea, in 1994, the M.S. degree in computer science from Stanford University, Stanford, CA, USA, in 1998, and the Ph.D. degree in computer science from the University of Pennsylvania, Philadelphia, PA, USA, in 2006.

He joined KAIST, Daejeon, South Korea, in 2008, where he is currently an Associate Professor with the Department of Computer Science. His current research interests include cyber-physical systems and real-time embedded systems.

Dr. Shin was a recipient of the Best Paper Award of RTSS in 2003 and 2012, the Best Student Paper Award of RTAS in 2011, and the Best Paper Runner-Ups of ECRTS and RTSS in 2008. He is currently an Editorial Board member of the *Journal of Computing Science and Engineering*. He has been the Program Co-Chair of RTCSA and has served various Program Committees in real-time embedded systems, including RTSS, RTAS, and ECRTS.