Batch-MOT: Batch-Enabled Real-Time Scheduling for Multiobject Tracking Tasks

Donghwa Kang, Seunghoon Lee, Cheol-Ho Hong, *Member, IEEE*, Jinkyu Lee, *Senior Member, IEEE*, and Hyeongboo Baek, *Member, IEEE*

***Abstract*—Targeting a multiobject tracking (MOT) system with multiple MOT tasks, this article develops Batch-MOT, the first system design that achieves both (G1) timing guarantee and (G2) accuracy maximization, by utilizing *batch execution* that allows multiple deep neural network (DNN) executions to perform simultaneously in a single DNN inference resulting in significantly decreased execution time without accuracy loss. To this end, we propose an adaptable scheduling framework that allows run-time execution behaviors deviated from our base scheduling algorithm (i.e., nonpreemptive fixed-priority scheduling) without compromising G1. Based on the adaptable framework, we then develop 1) a run-time batching mechanism that finds and executes a batch set of MOT tasks and 2) a run-time idling mechanism that waits for the future releases of MOT tasks for batch execution. Both run-time mechanisms can achieve G1 and G2 without incurring high run-time overhead, as they systematically exploit the run-time execution behaviors allowed by the adaptive framework. Our evaluation conducted with a real-world data set demonstrates the effectiveness of Batch-MOT in improving tracking accuracy while providing a timing guarantee compared to the state-of-the-art real-time MOT system for multiple MOT tasks.**

***Index Terms*—Batch execution, multiobject tracking (MOT), real-time scheduling, timing guarantee.**

I. Introduction

As MODERN autonomous vehicles (AVs) are equipped with multiple cameras, they require performing multiple multiobject tracking (MOT) tasks under limited computing resources. Perception tasks, such as MOT are required to complete their execution before specified deadlines because AVs' safety-related functions for path planning and vehicle control heavily rely on the timely perception, e.g., determining the time-to-collision with pedestrians and cars ahead as extensively discussed in the previous studies [1], [2], [3], [4]. Additionally, it is widely acknowledged that low accuracy in the perception tasks also compromises the safety of AVs [2], [3]. Therefore, an MOT system with multiple MOT tasks must achieve two goals simultaneously for every MOT task: 1) (G1) timing guarantee and 2) (G2) accuracy maximization.

The deep neural network (DNN)-based MOT approaches are increasingly deployed in modern AVs [5], [6], [7], [8], [9]. However, it is challenging to fully utilize them in order to achieve G1 and G2 for an MOT system with multiple MOT tasks due to the inherent tradeoff between G1 and G2. A recent study developed a scheduling framework that provides different execution options by efficiently utilizing the control knob of processing either a full-size or a down-scaled input image, which is the only existing study that addresses both G1 and G2 for multiple MOT tasks [3]. However, this control knob has its tradeoff; ensuring G1 might compromise G2 by necessitating downscaled image processing.

To overcome the tradeoff between G1 and G2, we utilize *batch execution* for multiple MOT tasks, which, supported by modern DNN models, concurrently processes multiple inputs in one DNN inference reducing execution time without accuracy loss by optimizing GPU resources [4], [10], [11]. As shown in Fig. 1(a) of the experiment results for a GPU of Tesla V100 (comparable to the NVIDIA Orin system-on-chip (SoC) providing similar GPU capability for Tensor core operations [12]) with the state-of-the-art DNN model (i.e., YOLOX [13]), 1.0 time unit taken for processing 12 *full-size* (size of 672 × 672) images *individually* (one by one) decreases to 0.46 time unit with *batch execution* without accuracy loss (maintaining 41%). Notably, this processing time is even smaller than 0.74 time unit taken for processing 12 *down-scaled* (size of 256 × 256) images *individually* (one by one) with accuracy drop to 17.7%. This is because it maintains nearly the same DNN inference time (see ii. in Fig. 1(b) for 19–21 ms) until the GPU reaches resource saturation, which occurs when processing more than ten input images for batch execution. On the other hand, the execution times of the other parts (to be detailed in Section II) linearly increase with the number of input images in a batch. We conducted the same experiments on a GPU-enabled embedded board (i.e., NVIDIA

Manuscript received 4 August 2024; accepted 4 August 2024. This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) under Grant 2022R1A4A3018824 and Grant RS-2024-00438248; in part by the National Research and Development Program through the National Research Foundation of Korea (NRF) funded by Ministry of Science and ICT under Grant 2021M3H2A1038042. The work of Hyeongboo Baek supported by the 2024 Research Fund of the University of Seoul. This article was presented at the International Conference on Embedded Software (EMSOFT) 2024 and the ESWEEK-TCAD special issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding authors: Jinkyu Lee; Hyeongboo Baek.)*

Donghwa Kang is with the Department of Computer Science and Engineering, Incheon National University, Incheon 22012, South Korea.

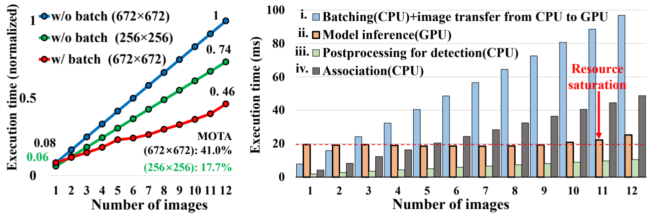
Seunghoon Lee and Jinkyu Lee are with the Department of Computer Science and Engineerning, Sungkyunkwan University, Jongno 03063, South Korea (e-mail: jinkyu.yi.3@gmail.com).

Cheol-Ho Hong is with the Department of Intelligent Semiconductor Engineering, Chung-Ang University, Dongjak 06974, South Korea.

Hyeongboo Baek is with the Department of Artificial Intelligence, University of Seoul, Dongdaemun 02504, South Korea (e-mail: hbbaek@uos.ac.kr).

This article has supplementary downloadable material available at https://doi.org/10.1109/TCAD.2024.3443002, provided by the authors.

Digital Object Identifier 10.1109/TCAD.2024.3443002



(a) (b)

Fig. 1. Execution times for different number of input images of an MOT system with YOLOX on a Tesla V100 GPU. (a) Total execution. (b) Decomposition of batch execution.

Xavier SoC [14]) and observed that it can accommodate two input images (see Fig. 6(d) for 36–38 ms) for a batch execution before the resource saturation.

Motivated by the shorter execution time by batch execution without accuracy loss, we target a set of MOT tasks, in which G1 is compromised by processing the original DNN workloads (i.e., full-size image) *individually*[1] but is not compromised by processing the reduced DNN workloads (i.e., the down-scaled image that compromises G2) *individually*. Then, we consider the individual execution with the reduced DNN workloads in default and aim at performing batch execution with the original DNN workloads from multiple MOT tasks as frequently as possible at run-time, such that both G1 and G2 are achieved; this entails the following challenges.

- C1: To determine the batch set to execute, we require an online mechanism to find feasible batch sets, along with run-time information on active MOT tasks, which results in significant run-time overhead; thus, a new scheduling framework with the *low run-time overhead* is essential.
- C2: While batch execution can expedite overall processing, it may delay specific tasks due to factors like priority inversion (to be detailed in Fig. 4 in Section V), necessitating a runtime mechanism (based on the answer to C1) with a schedulability test to ensure the timely task execution.
- C3: Since, any work-conserving scheduling cannot yield any batch execution under a situation where there is only one active task, we need a run-time idling mechanism (based on the answer to C1) that accelerates the batch execution for the situation while ensuring the timely execution of every task (based on the answer to C2).

In this article, we propose Batch-MOT, the first system design to achieve G1 and G2 by utilizing *batch execution* for multiple MOT tasks, which systematically tackles the chaining challenges C1–C3. Batch-MOT employs nonpreemptive fixed-priority scheduling (NPFP) as a base scheduling algorithm, in which each MOT task is executed nonpreemptively and the task priority is predefined.

To address C1, we analyse the offline schedulability test that provides timing guarantees under NPFP. Based on the analysis, we propose a new scheduling framework $\mathsf{NP}_{\mathsf{ADAPT}}$ (the nonpreemptive ADAPTable scheduling framework) that allows run-time execution behaviors to deviate from NPFP without compromising timing guarantees achieved by the offline schedulability test.

The strategy of $\mathsf{NP}_{\mathsf{ADAPT}}$ to reduce runtime overhead involves the *offline* identification of the amount of allowable run-time execution behavior deviations (denoted by Δ_k defined in Section IV-B) from NPFP for each MOT task, providing an interface for developing a *runtime* batching mechanism that incurs low runtime overhead without compromising timing guarantees.

As to C2, we develop a run-time batching mechanism $\mathsf{NPFP}^{\mathcal{B}}$ (NPFP with batch execution) based on $\mathsf{NP}_{\mathsf{ADAPT}}$. It finds a set of MOT tasks with a low run-time overhead, such that executing the set as a batch does not compromise the timely execution of any task. This is achieved by systematically exploiting the properties to be discussed in Section III and the run-time execution behaviors allowed by $\mathsf{NP}_{\mathsf{ADAPT}}$.

To address C3, we propose an advanced run-time batching mechanism with an idling scheme $\mathsf{NPFP}^{\mathcal{BI}}$ ($\mathsf{NPFP}^{\mathcal{B}}$ with idling), developed on top of $\mathsf{NPFP}^{\mathcal{B}}$. $\mathsf{NPFP}^{\mathcal{BI}}$ further utilizes the run-time execution behaviors allowed by $\mathsf{NP}_{\mathsf{ADAPT}}$ and determines the idling interval for each MOT task to wait for the future release(s) of the other MOT tasks to be executed as a batch, without incurring much run-time overhead. The relationship among $\mathsf{NP}_{\mathsf{ADAPT}}$, $\mathsf{NPFP}^{\mathcal{B}}$, and $\mathsf{NPFP}^{\mathcal{BI}}$ is described in Figure S.1 in the supplement [15].

We implemented Batch-MOT and evaluated it using an open MOT data set of the autonomous driving system. Our evaluation demonstrates that Batch-MOT exhibits higher tracking accuracy and lower run-time overhead without compromising timing guarantee, compared to the only existing study addressing both G1 and G2 for multiple MOT tasks [3].

We clarify our novelty and contribution along with an explanation of related work as follows.

1) To the best of our knowledge, Batch-MOT is the first study providing a *strict* timing guarantee for batch DNN execution of multiple (camera) tasks. Even extending our interest to general DNN beyond MOT, the existing studies address different problems from ours. That is, [2], [3], [11], [16], [17] do not deal with a timing guarantee for batch DNN execution; [4], [18] are designed for single-camera (i.e., single-task) systems; and [10], [19] aim at improving the overall FPS or minimizing the deadline miss ratio, therefore not addressing strict timing guarantees.
2) Batch-MOT enables the assurance of timing guarantees through a simple online test with low scheduling cost despite the complicated impact of batch DNN execution (both with/without idling) on the timing guarantee. This is achieved by a) the novel design of the scheduling frameworks $\mathsf{NP}_{\mathsf{ADAPT}}$, $\mathsf{NPFP}^{\mathcal{B}}$, and $\mathsf{NPFP}^{\mathcal{BI}}$ (to be detailed in Sections IV–VI, respectively) and b) the mathematical foundation of their timely correctness, both of which are highly challenging.

[1] If not compromised, the computing resource is sufficient for achieving G1 and G2 without any advanced technique, which is not the scope of this article.

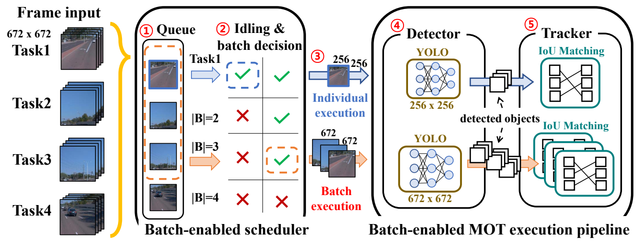


Fig. 2. Overview of Batch-MOT

II. Overview of Batch-MOT

As illustrated in Fig. 2, the core design features of Batch-MOT include the *batch-enabled MOT execution pipeline* and the *batch-enabled scheduler* to be detailed in this section. The MOT execution pipeline and scheduler are implemented as separate threads, and they communicate with each other by exchanging messages through the shared memory. The workflow of Batch-MOT is as follows. Each input frame of the MOT tasks is forwarded to a ready queue (① in Fig. 2). Once the batch-enabled scheduler determines the idling time and batch size of MOT tasks (②), some MOT tasks in the ready queue are combined into a batch or no batch is constructed according to the idling and batch decision (③). Then, detection (④) and association (⑤) are conducted sequentially for either batch or individual execution.

A. Batch-Enabled MOT Execution Pipeline

In the batch-enabled MOT execution pipeline, the *front-end* DNN-based detector identifies the position and size of each object's bounding box in the input frame and sends the detection information to the *back-end* tracker. The tracker then conducts an association to match each detected object with one of the existing objects in the previous frame (called tracklet) based on the intersection over union-based (IoU-based) matching and updates the tracking information for the matched tracklet.

For the detector, Batch-MOT adopts any existing standalone DNN models (e.g., the YOLO series [13], [20]) that can accept variable input image sizes determining the tracking accuracy as shown in Fig. 1(a). The batch MOT execution pipeline supports two types of execution: 1) *individual* and 2) *batch* execution. For an input image of the size of 672×672, the individual execution scale-downs the input image to the size of 256×256. Then, it conducts the detection of the MOT tasks and shows decreased execution time at the expense of sacrificing the tracking accuracy. In a batch execution, multiple input images with the original size of 672×672 are combined in a batch for DNN inference, and it is transferred from the CPU to GPU memory [i. in Fig. 1(b)]. Then, the DNN inference is conducted on the GPU to detect candidate objects [ii) in Fig. 1(b)], and the postprocessing, such as nonmaximum suppression (NMS) [20] is performed on the CPU to extract high-confident objects among detected candidates [iii) in Fig. 1(b)]. As the tracker uses IoU-based matching, it compares the position and size of tracklets with the objects detected in the current frame on a one-to-one basis and matches two objects whose size of overlapping region is greater than a given threshold on the CPU. In the case of batch execution, after detection is performed for multiple MOT tasks, the associations for the batch are then performed sequentially on the CPU [iv) in Fig. 1(b)]; however, if the time cost for interprocess communication (IPC) on the platform is relatively low compared to the execution time of an association, the associations can be executed in parallel across multiple CPUs using multiprocessing, which decreases the overall execution time.

B. Batch-Enabled Scheduler

Batch-MOT supports a thread-level scheduler invoked when an MOT task is released or completed. The proposed batch-enabled scheduler operates as a background daemon and communicates with the MOT execution pipeline through the shared memory. To make Batch-MOT capable of addressing C1–C3, the batch-enabled scheduler is designed as follows.

To tackle C1–C3, Batch-MOT needs to implement a run-time batch decision mechanism that does not compromise timing guarantees while maintaining the low run-time overhead. This is challenging because batch execution affects the behavior of multiple tasks, including i) the target task; ii) the lower priority tasks; and iii) the higher priority tasks. In other words, the batch execution of given $\mathcal{B}$ can change the execution of i), ii), and/or iii), and the impact of one is different from that of the others. Considering all the possible execution variations of batch executions and their influences on the other tasks at every scheduling decision may cause prohibitively high run-time overhead, which has not been addressed in the previous study [3].

To overcome the challenge, we first analyse the underlying principle of our base scheduling algorithm NPFP and its schedulability analysis that judges the timing guarantee of the given task τ_k by considering i), ii), and iii) to be detailed in Section IV-A. We then develop a new scheduling framework, $\mathsf{NP}_{\mathsf{ADAPT}}$, which enables the run-time execution behaviors of i), ii), and iii) deviated from NPFP while preserving the schedulability guaranteed under NPFP, by associating the run-time execution behaviors with the schedulability test to be detailed in Section IV-B. By using the run-time execution behaviors with the properties to be discussed in Section III, the scheduler finds and executes schedulable batch sets under work-conserving scheduling (addressing C1 and C2 to be detailed in Section V) and beyond work-conserving scheduling (addressing C1 and C3, to be detailed in Section VI), with the low run-time overhead.

III. System Model

We consider an MOT system with multiple DNN-based MOT tasks $\tau = \{\tau_i\}_{i=1}^{n}$ [3] on a platform equipped with multiple CPUs and a single GPU. As an input video frame is provided periodically, an MOT task τ_i is considered a periodic real-time task with a timing constraint. That is, an MOT task τ_i invokes a series of jobs J_i, each separated by exactly T_i time units; once a job of τ_i is released at t, it should finish its execution no later than its deadline $t + T_i$. The period of each task is not necessarily the same, which makes it possible to address the situation where the frame rate of each camera varies based on its intended use (e.g., side-facing cameras typically operate at lower rates while forward-facing cameras operate at higher rates [1]); of course, our task model also accommodates a set of tasks with the same period. A job is said to be *active* at t, if it has remaining execution at t. Let $\tau(t)$ denote a set of tasks, each of whose job is active at t. Let $r_i(t)$ denote the earliest job release time of τ_i after t. Since, each task is strictly periodic, it is possible to know $r_i(t)$ at t and indicate the earliest job deadline of τ_i after t. Let $\mathtt{LP}(\tau_k)$ and $\mathtt{HP}(\tau_k)$ denote a set of tasks whose priority is lower and higher than τ_k, respectively. Notations are summarized in Table S.1 in the supplement.

An MOT task set τ is said to be *schedulable* under a target scheduling framework if every job invoked by tasks in τ does not miss its deadline when the framework schedules τ. Since, there is at most one active job of $\tau_i \in \tau$ at any time, we use a task τ_i and a job of τ_i (denoted by J_i) interchangeably when no ambiguity arises. As presented, we aim to achieve G1 and G2 for every MOT task. Let C_i denote the worst-case execution time (WCET) of τ_i when each performs *individual* execution (as opposed to *batch* execution).

To schedule a set of MOT tasks, we decide to apply the following two policies. First, we enforce nonpreemptiveness between the detection and association of each job; that is, once a job of τ_i starts its execution, it sequentially performs the execution of its detection and association subjobs without any preemption. Second, we disallow individual MOT tasks to be executed in parallel (if not a part of a batch), while the associations of MOT tasks in a batch can be performed simultaneously on multiple CPUs depending on the time cost of IPC mentioned in Section II. The two policies not only reduce the run-time scheduling overhead but also significantly lower the complexity of considering various run-time scenarios that incur different interference/blocking; therefore, the policies make it possible to ensure offline timing guarantees through a simple online test with low scheduling cost to be developed in Sections IV–VI.

In this article, we process a down-scaled image (i.e., 256×256) as the *individual* execution of each job, while we do a full-size image (i.e., 672×672) as the *batch* execution of a group of jobs. Let $\mathcal{B}$ denote a set of tasks whose jobs will be executed as a batch, and let $C_{\mathcal{B}}$ denote the WCET of $\mathcal{B}$, where $|\mathcal{B}| \geq 2$. We take the measurement-based approach to derive the WCET of MOT tasks, using the experiment setup in Section VII, with an in-depth discussion provided in Section VIII.

In this article, we use the following properties of batch execution.

P1: $C_{\mathcal{B}} \geq \max_{\tau_i \in \mathcal{B}} C_i$;

P2: $C_{\mathcal{B}} \leq \sum_{\tau_i \in \mathcal{B}} C_i$; and

P3: $C_{\mathcal{B}} \leq C_{\mathcal{B}'}$, if $\mathcal{B} \subset \mathcal{B}'$.

Batch execution of multiple MOT tasks dramatically shortens total inference latency by reducing the number of GPU invocations. That is, *individual* execution requires as many invocations as the number of given MOT tasks, while *batch* execution performs inference with one invocation. Since, the inference latency through a single GPU invocation increases monotonically according to the DNN workload of the invocation, P1 holds generally. Likewise, since DNN workload of $\mathcal{B}$ will be increased if we add more job(s) to $\mathcal{B}$, P3 holds generally. Apart from P1 and P3, which typically holds, P2 holds under the following condition: the benefit of reducing the number of GPU invocations outweighs the increasing workload (from a smaller total DNN workload of multiple *individual* executions to a larger DNN workload of a single *batch* execution). To satisfy the condition, we need to deploy a detector that offers high optimization of batch execution. In this article, we target the state-of-the-art detectors optimized for batch execution that ensure P2 is met (e.g., YOLOX [13] illustrated in Fig. 1, YOLOv5 [20], Faster-RCNN [21], and others with varying sizes for both the downscaled and full-size input images).

IV. Developing Adaptable Scheduling Framework

A. Base Scheduling Algorithm NPFP

In this article, we employ NPFP [3], [22] as a base scheduling algorithm. As explained in Section III, once a job of τ_i starts its execution, it sequentially performs the execution of its detection and association subjobs without any preemption (by NP). Also, each task's priority is predefined, and each job inherits its invoking task's priority (by FP). Whenever there is at least one active job while the computing system is idle, NPFP selects the highest-priority active job and starts its execution. NPFP is work conserving, meaning that the computing system cannot be idle if there is at least one active job; also, the vanilla NPFP does not allow any batch execution.

The schedulability of a set of MOT tasks under NPFP is guaranteed by the next lemma [3], [23], which is a sufficient but not necessary schedulability test.

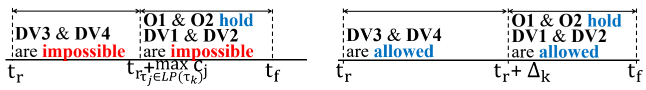
Lemma 1 (From [3], [23]): An MOT task set τ is schedulable by NPFP, if $R_k \leq T_k$ holds for every $\tau_k \in \tau$, where R_k (i.e., the response time of τ_k) can be calculated as follows. $R_k(x+1)$ is calculated by (1) sequentially with $x = 0, 1, 2, \ldots$, starting from $R_k(0) = C_k + \sum_{\tau_h \in \mathtt{HP}(\tau_k)} C_h + \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j$, until $R_k(x+1) = R_k(x)$ (implying $R_k = R_k(x)$) or $R_k(x+1) > T_k$ (implying no bounded R_k)

$$R_k(x+1) = C_k + \sum_{\tau_h \in \mathtt{HP}(\tau_k)} \left\lceil \frac{R_k(x)}{T_h} \right\rceil \cdot C_h + \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j. \quad (1)$$

Proof: Here, we summarize the proof in [3] and [23]. Consider the following situation.

1) The first job of τ_k and every first job of tasks whose priority is higher than τ_k are released at t_0.
2) A job of a task whose C_i is the largest among tasks whose priority is lower than τ_k is released right before t_0.
3) The following jobs of τ_k and those of tasks whose priority is higher than τ_k are released periodically.

It was proven that one of the jobs of τ_k released under the situation (but not necessarily the first job of τ_k released at t_0) yields the largest response time of τ_k [22].



(a) (b)

Fig. 3. Properties of NPFP and $\mathsf{NP}_{\mathsf{ADAPT}}$. (a) NPFP (b) $\mathsf{NP}_{\mathsf{ADAPT}}$.

Then, it is trivial that the response time of the first job of τ_k under the situation is upper-bounded by $R_k(x)$ that satisfies (1). The proof of [3, Lemma 1] proves that the response time of the $(x+1)^{th}$ job of τ_k cannot be larger than that of the x^{th} job (where $x \geq 1$), if $R_k \leq T_k$ holds with (1), meaning there is no self-pushing phenomenon issue [22] for τ_k if $R_k \leq T_k$ holds with (1). ∎

B. Adaptable Scheduling Framework $\mathsf{NP}_{\mathsf{ADAPT}}$

Once Lemma 1 deems τ schedulable, we can guarantee timely execution of τ scheduled by NPFP. However, the accuracy of MOT tasks in τ scheduled by NPFP cannot be maximized, because every job under NPFP performs individual execution with a down-scaled image (as opposed to batch execution of multiple jobs with a full-size image). Therefore, targeting τ deemed schedulable under the vanilla NPFP (that does not employ batch execution), we want to maximize the MOT accuracy by performing batch execution as much as possible without compromising the schedulability.

However, a batch execution of a set of multiple jobs easily compromises each job's timely execution achieved by the individual execution of corresponding jobs. For example, if a higher-priority job of τ_i and a lower-priority job of τ_j are executed as a batch, the execution time of the batch could be larger than that of the job of τ_i solely (by P1), which may yield the deadline miss of the job of τ_i. Therefore, we need to identify which run-time execution behaviors (caused by batch execution) that deviate from NPFP do not compromise the schedulability.

To establish boundaries of run-time execution behavior deviation (e.g., execution time increment due to batch execution, intentional idling for batch execution) without compromising the schedulability, we target a task set that satisfies Lemma 1 and analyse how Lemma 1 guarantees the schedulability under NPFP. To this end, we focus on a job of τ_k that is released and finished at t_r and t_f, respectively. First, by the property of the nonpreemptiveness of NPFP, a lower-priority job can block the execution of a higher-priority job *only when* the former starts its execution before the release of the latter. As addressed by the third term of the RHS of (1), the following holds in $[t_r + \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j, t_f)$ under NPFP, illustrated in Fig. 3(a).

O1: Except the execution of the job of τ_k and jobs with higher priority than τ_k, any other run-time behavior (e.g., other jobs' execution, the system idling) is disallowed.

Second, after the lower-priority blocking, the only possible execution behaviors that affect the schedulability of the job of τ_k are the execution of the job of τ_k itself and jobs with higher priority than τ_k. As addressed by the first two terms of the RHS of (1), the following holds in $[t_r + \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j, t_f)$ under NPFP if Lemma 1 holds. This is illustrated in Fig. 3(a).

O2: The amount of execution of the job of τ_k and jobs with higher priority than τ_k does not exceed $C_k + \sum_{\tau_h \in \mathtt{HP}(\tau_k)} \lceil ([t_f - t_r]/T_h) \rceil \cdot C_h$.

Considering the two properties, we define a class of the nonpreemptive scheduling algorithms with the minimum requirements, which 1) allows run-time execution behavior deviated from NPFP to be potentially utilized for batch execution and 2) does not compromise the schedulability under NPFP guaranteed by Lemma 1.

Definition 1: We define $\mathsf{NP}_{\mathsf{ADAPT}}$ associated with given $\{\Delta_k {\geq} 0\}_{\tau_k \in \tau}$ (the nonpreemptive ADAPTable scheduling framework), as any nonpreemptive scheduling algorithm in which every job of $\tau_k \in \tau$ (that is released and finished at t_r and t_f, respectively) satisfies the following features, which are illustrated in Fig. 3(b).

F1: In $[t_r + \Delta_k, t_f)$, O1 holds.

F2: In $[t_r + \Delta_k, t_f)$, O2 holds.

Since, F1 and F2 are the only requirements, $\mathsf{NP}_{\mathsf{ADAPT}}$ with given $\{\Delta_k\}$ can accommodate the following possible run-time execution behaviors deviated from NPFP *as long as F1 and F2 hold*. Recall that C_i for each τ_i is the WCET when its job performs individual execution with a down-scaled image (as opposed to batch execution of multiple jobs with a full-size image).

1) In $[t_r + \Delta_k, t_f)$, (DV1) the job of τ_k may execute for more than C_k.
2) In $[t_r + \Delta_k, t_f)$, (DV2) a job of $\tau_h \in \mathtt{HP}(\tau_k)$ may execute for more than C_h.
3) In $[t_r, t_r + \Delta_k)$, (DV3) a job of $\tau_j \in \mathtt{LP}(\tau_k)$ executes for more than C_j.
4) In $[t_r, t_r + \Delta_k)$, (DV4) the computing system becomes idle even though there is an active job, meaning that $\mathsf{NP}_{\mathsf{ADAPT}}$ is not work conserving.

The above run-time execution behaviors will be used for the run-time batching mechanism to be explained in Sections V and VI. We would like to emphasize that DV1–DV4 are run-time execution behaviors deviated from NPFP, as NPFP does not allow them in the corresponding intervals; in other words, NPFP disallows DV1 and DV2 in $[t_r + \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j, t_f)$, and DV3 and DV4 in $[t_r, t_r + \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j)$.

Considering that NPFP satisfies O1 and O2 in $[t_r + \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j, t_f)$ if Lemma 1 holds, the following property holds: for τ that is deemed schedulable by Lemma 1, NPFP belongs to $\mathsf{NP}_{\mathsf{ADAPT}}$ associated with $\{\Delta_k = \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j\}$. From F1 and F2 for $\mathsf{NP}_{\mathsf{ADAPT}}$, we can easily derive the schedulability analysis of $\mathsf{NP}_{\mathsf{ADAPT}}$, by replacing $\max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j$ with Δ_k in Lemma 1.

Theorem 1: An MOT task set τ is schedulable by $\mathsf{NP}_{\mathsf{ADAPT}}$ associated with given $\{\Delta_k\}$, if $R_k \leq T_k$ holds for every $\tau_k \in \tau$, where R_k (i.e., the response time of τ_k) can be calculated as follows. $R_k(x+1)$ is calculated by (2) sequentially with $x = 0, 1, 2, \ldots,$, starting from $R_k(0) = C_k + \sum_{\tau_h \in \mathtt{HP}(\tau_k)} C_h + \Delta_k$, until $R_k(x+1) = R_k(x)$ (implying $R_k = R_k(x)$) or $R_k(x+1) >$

T_k (implying no bounded R_k)

$$R_k(x+1) = C_k + \sum_{\tau_h \in \mathrm{HP}(\tau_k)} \left\lceil \frac{R_k(x)}{T_h} \right\rceil \cdot C_h + \Delta_k. \qquad (2)$$

Proof: Suppose that, a job of τ_k whose release time is t_r does not finish its execution until $t_r + R_k$, although R_k satisfies (2). First, we consider that the amount of execution of τ_k and its higher-priority tasks in $[t_r + \Delta_k, t_r + R_k)$ is not larger than $C_k + \sum_{\tau_h \in \mathrm{HP}(\tau_k)} \lceil (R_k/T_h) \rceil \cdot C_h$. From (2), $R_k - \Delta_k = C_k + \sum_{\tau_h \in \mathrm{HP}(\tau_k)} \lceil (R_k/T_h) \rceil \cdot C_h$ holds. Therefore, if the amount of execution of τ_k and its higher-priority tasks in $[t_r + \Delta_k, t_r + R_k)$ is not larger than $C_k + \sum_{\tau_h \in \mathrm{HP}(\tau_k)} \lceil (R_k/T_h) \rceil \cdot C_h$, the supposition contradicts F1. Second, we consider that the amount of execution of τ_k and its higher-priority tasks in $[t_r + \Delta_k, t_r + R_k)$ is larger than $C_k + \sum_{\tau_h \in \mathrm{HP}(\tau_k)} \lceil (R_k/T_h) \rceil \cdot C_h$, which immediately contradicts F2. Therefore, the supposition always contradicts. ■

Provided that $\Delta_k \geq \max_{\tau_j \in \mathrm{LP}(\tau_k)} C_j$, no execution of τ_j can occur within the interval $[t_r + \Delta_k, t_f)$, where t_r and t_f represent the release and finishing times of τ_k, respectively. This condition ensures the sustainability property with respect to $\{C_i\}$.

In the next section, we will develop a run-time batching mechanism by utilizing the capability of $\mathsf{NP}_{\mathsf{ADAPT}}$ in achieving the schedulability even in the presence of the run-time execution behavior deviated from NPFP (as long as F1 and F2 are satisfied). To this end, we will deploy the largest Δ_k for $\mathsf{NP}_{\mathsf{ADAPT}}$ to accommodate a longer blocking period due to batch execution or a longer idling for batch execution to be performed in the future. Let Δ_k^* denote the largest Δ_k that does not compromise the schedulability of τ_k in Theorem 1, and let R_k^* denote R_k with Δ_k^*. We can easily verify that if τ is deemed schedulable by Lemma 1, $\Delta_k^* \geq \max_{\tau_j \in \mathrm{LP}(\tau_k)} C_j$ holds for every $\tau_k \in \tau$.

Offline Time-Complexity: We can find $\Delta_k^* \in [0, T_k - C_k]$ using the binary search. Hence, the time complexity to find Δ_k^* and R_k^* for every $\tau_k \in \tau$ using Theorem 1 is $O(n^2 \cdot \log(n) \cdot \max(T_k))$, which is affordable as it is performed offline.

V. $\mathsf{NPFP}^{\mathcal{B}}$: Enabling Run-Time Batching

As C1 and C2 in Section I indicate, utilizing batch execution necessitates a mechanism that efficiently finds a batch of jobs to be executed *at run-time* without compromising timing guarantee. Therefore, this section develops a run-time mechanism that achieves the following goals to address C1 and C2 in Section I, respectively.

1) Perform batch execution as frequently as possible (for high accuracy) while minimizing the run-time complexity.
2) Do not compromise the schedulability of τ under NPFP, guaranteed by Lemma 1.

To this end, we develop $\mathsf{NPFP}^{\mathcal{B}}$ (NPFP with batch execution) associated with given $\{\Delta_k\}$. We address the second goal by making $\mathsf{NPFP}^{\mathcal{B}}$ follow F1 and F2 (implying $\mathsf{NPFP}^{\mathcal{B}}$ with $\{\Delta_k\}$ belongs to $\mathsf{NP}_{\mathsf{ADAPT}}$ with $\{\Delta_k\}$). Then, the remaining step is how to design a run-time batching mechanism that addresses the first goal while satisfying F1 and F2.

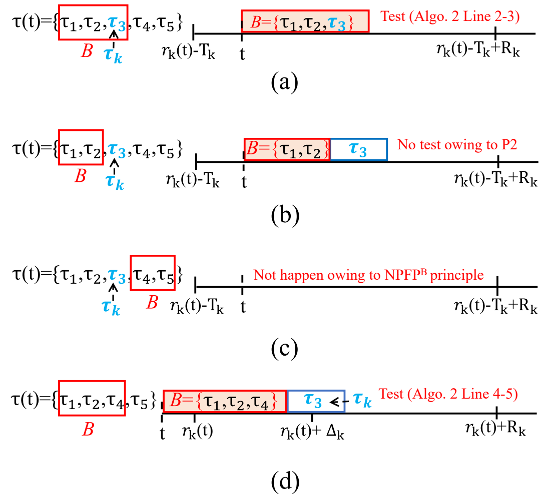


Fig. 4. Four batch execution cases for a set of tasks $\{\tau_1, \tau_2, \tau_3(=\tau_k), \tau_4, \tau_5\}$; the smaller index, the higher priority. (a) Case 1: $\tau_k \in \tau(t)$ and $\tau_k \in \mathcal{B}$. (b) Case 2: $\tau_k \in \tau(t)$, $\tau_k \notin \mathcal{B}$, and $\tau_h \in \mathrm{HP}(\tau_k)$. (c) Case 3: $\tau_k \in \tau(t)$, $\tau_k \notin \mathcal{B}$, and $\tau_h \in \mathrm{LP}(\tau_k)$. (d) Case 4: $\tau_k \notin \tau(t)$ (and therefore $\tau_k \notin \mathcal{B}$).

As a first step to develop $\mathsf{NPFP}^{\mathcal{B}}$ associated with given $\{\Delta_k\}$, we investigate how each batch execution affects the schedulability under NPFP guaranteed by Lemma 1. From now on, we interpret a run-time execution behavior deviated from NPFP due to batch execution as a change of WCET (as well as the actual execution time) of the highest-priority task in the batch; therefore, the priority of a batch execution inherits the priority of the highest-priority task in the batch. For example, if a higher-priority job of τ_i and a lower-priority job of τ_j are executed as a batch, we regard this situation as an increase of WCET of the job of τ_i from C_i to $C_{\mathcal{B}}$ where $\mathcal{B} = \{\tau_i, \tau_j\}$.

Consider τ deemed schedulable by Lemma 1. Suppose we schedule τ by NPFP until t, but we are going to execute a set of jobs as a batch (denoted by $\mathcal{B}$) at t. We investigate how the schedulability of a job J_k of the task $\tau_k \in \tau$ is affected differently according to the following four cases, which are illustrated in Fig. 4 with a task set $\tau = \{\tau_1, \tau_2, \tau_3(=\tau_k), \tau_4, \tau_5\}$, in which a smaller task index implies a higher priority. Let τ_h denote the highest-priority task among tasks in $\mathcal{B}$; recall that $\tau(t)$ is a set of tasks, each of whose job is active at t, and $r_k(t)$ is the earliest job release time of τ_k after t.

1) *Case 1 of $\tau_k \in \tau(t)$ and $\tau_k \in \mathcal{B}$:* The WCET of J_k is changed from C_k to $C_{\mathcal{B}}$, e.g., $\mathcal{B} = \{\tau_1, \tau_2, \tau_3(=\tau_k)\}$ in Fig. 4(a).
2) *Case 2 of $\tau_k \in \tau(t)$, $\tau_k \notin \mathcal{B}$, and $\tau_h \in HP(\tau_k)$:* The longest time for a job of τ_h to delay the execution of J_k is changed from C_h to $C_{\mathcal{B}}$, e.g., $\mathcal{B} = \{\tau_1, \tau_2\}$ in Fig. 4(b).
3) *Case 3 of $\tau_k \in \tau(t)$, $\tau_k \notin \mathcal{B}$, and $\tau_h \in LP(\tau_k)$:* J_k experiences an additional delay from a lower-priority job of τ_h for up to $C_{\mathcal{B}}$, which does not occur under NPFP, e.g., $\mathcal{B} = \{\tau_4, \tau_5\}$ in Fig. 4(c).
4) *Case 4 of $\tau_k \notin \tau(t)$ (therefore $\tau_k \notin \mathcal{B}$):* The longest time for a job of τ_h to delay the execution of J_k (to be released at $r_k(t) > t$) is changed from $\max(0, t + C_h - r_k(t))$ to $\max(0, t + C_{\mathcal{B}} - r_k(t))$, e.g., $\mathcal{B} = \{\tau_1, \tau_2, \tau_4\}$ in Fig. 4(d).

To make $\mathsf{NPFP}^{\mathcal{B}}$ associated with given $\{\Delta_k\}$ preserve schedulability even in the presence of batch execution for Cases 1–4, our basic design principle for the run-time mechanism of $\mathsf{NPFP}^{\mathcal{B}}$ is as follows.

We enforce the prioritization policy of FP on batch execution. To this end, we disallow the batch execution of $\mathcal{B}$ at t, if there is an active job of $\tau_k \notin \mathcal{B}$ whose priority is higher than the lowest-priority task in $\mathcal{B}$. The principle has three distinct advantages as follows.

DP1: To make the schedule under $\mathsf{NPFP}^{\mathcal{B}}$ as similar as possible to that under the base scheduling algorithm NPFP,

DP2: To eliminate Case 3, which a) not only reduces a burden to check the schedulability affected by given batch execution b) but also helps to comply with F1 by preventing a lower-priority job from executing in the interval of interest of F1 (i.e., $[t_r + \Delta_k, t_f)$).

DP3: To narrow down the number of possible choices of the set of tasks to be executed as a batch, i.e., from $2^{|\tau(t)|} - 1$ to $|\tau(t) - 1|$, which significantly reduces the run-time overhead for checking different batch candidates as well as the offline measurement/analysis overhead for obtaining $C_{\mathcal{B}}$ for different $\mathcal{B}$.

Under the design principle, we handle Cases 1–4 for a given batch execution. For each case, we either derive a condition for the batch execution not to compromise the schedulability (for Cases 1 and 4) or verify that the batch execution cannot compromise the schedulability (for Cases 2 and 3), both of which are achieved by satisfying F1 and F2.

1) *Case 1 of $\tau_k \in \tau(t)$ and $\tau_k \in \mathcal{B}$:* If $t + C_{\mathcal{B}} \le r_k(t) - T_k + R_k$ holds, F2 is satisfied and the job of τ_k active at t does not miss its deadline,[2] as exemplified in Fig. 4(a).
2) *Case 2 of $\tau_k \in \tau(t)$, $\tau_k \notin \mathcal{B}$, and $\tau_h \in \mathtt{HP}(\tau_k)$:* The longest time for jobs of tasks in $\mathcal{B}$ (at most one job per task) to be executed is decreased from $\sum_{\tau_h \in \mathcal{B}} C_h$ (in the $\sum_{\tau_h \in \mathtt{HP}(\tau_k)} \lceil (R_k(x)/T_h) \rceil \cdot C_h$ term in the RHS of (1)) to $C_{\mathcal{B}}$ by P2, which cannot compromise the schedulability of J_k that is active at t, as exemplified in Fig. 4(b).
3) *Case 3 of $\tau_k \in \tau(t)$, $\tau_k \notin \mathcal{B}$, and $\tau_h \in \mathtt{LP}(\tau_k)$:* This case does not occur under the design principle DP2, as illustrated in Fig. 4(c).
4) *Case 4 of $\tau_k \notin \tau(t)$ (and Therefore $\tau_k \notin \mathcal{B}$):* If $t + C_{\mathcal{B}} \le r_k(t) + \Delta_k$ holds, the batch execution of $\mathcal{B}$ does not violate F1 and does not affect F2, as exemplified in Fig. 4(d).

The formal proof of the conditions/statements in Cases 1–4 will be in the proof of Theorem 2.

Using Cases 1–4, Algorithm 2 presents $\mathsf{NPFP}^{\mathcal{B}}$ associated with $\{\Delta_k\}$, which is performed at t at which there is at least one active job while the computing system is ready to work. After defining $\mathcal{B}(n)$ as a set of the n highest-priority tasks among $\tau(t)$ in line 1, we check whether there are multiple active jobs (i.e., two or more tasks in $\tau(t)$) in line 2. We find the largest x, such that $\mathcal{B}(x)$ does not compromise the schedulability of all the other jobs, by testing schedulability test for online batching

[2]Recall that R_k is the response time of τ_k calculated by Theorem 1 for given Δ_k.

Algorithm 1 $\mathtt{STOB}(t, \tau(t), \mathcal{B})$

1: **for** $\tau_k \in \tau$ **do**
2:   **if** $\tau_k \in \tau(t)$ and $\tau_k \in \mathcal{B}$ **then**
3:     **if** $t + C_{\mathcal{B}} > r_k(t) - T_k + R_k$ **then** *return* unschedulable
4:   **else if** $\tau_k \notin \tau(t)$ **then**
5:     **if** $t + C_{\mathcal{B}} > r_k(t) + \Delta_k$ **then** *return* unschedulable
6:   **end if**
7: **end for**
8: *return* schedulable

Algorithm 2 $\mathsf{NPFP}^{\mathcal{B}}$ Scheduling Algorithm

At t, at which a job is finished while there is at least one active job, or at which at least one job is released while the system is idle,

1: Let $\mathcal{B}(n)$ denote $\{\tau_i(t)\}_{i=1}^{n}$, where $\tau_n(t)$ denotes the n^{th} highest-priority task in the set of active tasks at t (i.e., $\tau(t)$) for $1 \le n \le |\tau(t)|$.
2: **if** $|\tau(t)| \ge 2$ **then**
3:   Find the largest batch set $\mathcal{B}(x)$ for $2 \le x \le |\tau(t)|$ such that $\mathtt{STOB}\big(t, \tau(t), \mathcal{B}(x)\big)$ in Algorithm 1 returns schedulable, using *binary search*; if such $\mathcal{B}(x)$ exists, execute a set of active jobs invoked by tasks in $\mathcal{B}(x)$ as a batch, and *return*.
4: **end if**
5: Execute the highest-priority active job, and *return*.

($\mathtt{STOB}$) $(t, \tau(t), \mathcal{B}(x))$ in Algorithm 1 (in Line 3); we will detail Algorithm 1, including why it is possible to apply the binary search. If such $\mathcal{B}(x)$ exists, execute a set of active jobs invoked by tasks in $\mathcal{B}(x)$ as a batch (in line 3). Otherwise (or there is only one active job at t), execute the highest-priority job (in line 5), which is the same as NPFP.

For a given $\mathcal{B}$, $\mathtt{STOB}$ in Algorithm 1 checks whether every task τ_k satisfies the conditions in Cases 1 and 4. Lines 2 and 3 correspond Case 1, while lines 4 and 5 correspond Case 4. Note that, by the statements of Cases 2 and 3, we do not need to check the cases for schedulability. Now, we present why it is possible to apply the binary search to find the largest $\mathcal{B}(x)$ in line 3 of Algorithm 2.

Lemma 2: Recall $\mathcal{B}(x)$ in line 1 of Algorithm 2. If $\mathtt{STOB}(t, \tau(t), \mathcal{B}(x{+}1))$ in Algorithm 1 returns schedulable, then $\mathtt{STOB}(t, \tau(t), \mathcal{B}(x))$ also returns schedulable.

Proof: Since, $\mathcal{B}(x) \subset \mathcal{B}(x{+}1)$ holds, $C_{\mathcal{B}(x)} \le C_{\mathcal{B}(x+1)}$ holds by P3 in Section III. This implies that the opposite conditions in lines 3 and 5 of Algorithm 1, respectively, satisfy $r_k(t) - T_k + R_k \ge t + C_{\mathcal{B}(x+1)} \ge t + C_{\mathcal{B}(x)}$ and $r_k(t) + \Delta_k \ge t + C_{\mathcal{B}(x+1)} \ge t + C_{\mathcal{B}(x)}$. Therefore, the lemma holds. ■

As we designed, $\mathsf{NP}_{\mathsf{ADAPT}}$ subsumes $\mathsf{NPFP}^{\mathcal{B}}$ as follows.

Theorem 2: For τ with $\{\Delta_k \ge \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j\}$ that is deemed schedulable by Theorem 1, $\mathsf{NPFP}^{\mathcal{B}}$ associated with $\{\Delta_k\}$ belongs to $\mathsf{NP}_{\mathsf{ADAPT}}$ associated with $\{\Delta_k\}$.[3]

Proof: Suppose that a job of τ_k (denoted by J_k) scheduled by $\mathsf{NPFP}^{\mathcal{B}}$ associated with $\{\Delta_k\}$ is released and finished at t_r and t_f, respectively. We prove that J_k always satisfies F1 and F2 of Definition 1, which proves the theorem.

First, we check whether F1 is satisfied. Since, $\mathsf{NPFP}^{\mathcal{B}}$ is work conserving, it suffices to check (F1′) whether there is no

[3]Since Theorem 1 with $\Delta_k = \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j$ is equivalent to Lemma 1, τ deemed schedulable by Theorem 1 with $\Delta_k \ge \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j$ is also deemed schedulable by Lemma 1 (i.e., NPFP-schedulable).

execution of lower-priority jobs in $[t_r + \Delta_k, t_f)$. We consider four cases.

(Case F1a): If a lower-priority job of τ_j starts its execution at t $(< t_r)$ as individual execution, it will finish its execution no later than $t + C_j$ $(< t_r + C_j)$. Therefore, as long as $\Delta_k \geq \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j$ holds, F1$'$ holds.

(Case F1b): If a batch (denoted by $\mathcal{B}$), including a lower-priority job of τ_j starts its execution at t $(< t_r)$, it will finish its execution no later than $t + C_{\mathcal{B}}$ $(< t_r + C_{\mathcal{B}})$. By line 5 of Algorithm 1, $t + C_{\mathcal{B}} \leq t_r + \Delta_k$ holds, meaning that F1$'$ holds.

(Case F1c): If a lower-priority job of τ_j starts its execution at t $(\geq t_r)$ although J_k is not finished until t, it violates line 5 of Algorithm 2.

(Case F1d): If a batch (denoted by $\mathcal{B}$), including a lower-priority job of τ_j starts its execution at t $(\geq t_r)$, we consider two subcases: (i) $\mathcal{B}$'s priority is lower than τ_k, and ii) otherwise. Recall that $\mathcal{B}$ has the priority of the highest-priority task in $\mathcal{B}$; so, the entire execution of $\mathcal{B}$ has equal or higher priority than τ_k. Therefore, i) violates line 5 of Algorithm 2, and ii) does not violate F1 since it is regarded as an execution whose priority is not lower than τ_k.

Second, we check whether F2 is satisfied with four cases.

(Case F2a): We consider there is no batch execution in $[t_r, t_f)$. Similar to the proof of Lemma 1 for NPFP, the amount of execution of τ_k and its higher-priority tasks in $[t_r + \Delta_k, t_f)$ is maximized when all the jobs of τ_k and its higher-priority tasks are released at t_r. In this worst-case situation, F2 trivially holds.

(Case F2b): We consider a batch (denoted by $\mathcal{B}$) starts its execution at t $(< t_r)$ (and therefore $\mathcal{B}$ cannot include τ_k). Then, the batch execution will finish its execution no later than $t + C_{\mathcal{B}}$ $(< t_r + C_{\mathcal{B}})$. By line 5 of Algorithm 1, $t + C_{\mathcal{B}} \leq t_r + \Delta_k$ holds, meaning that the batch execution cannot contribute to higher-priority execution in $[t_r + \Delta_k, t_f)$.

(Case F2c): We consider a batch (denoted by $\mathcal{B}$) that does not include τ_k starts its execution at t $(\geq t_r)$. Then, the WCET of $\mathcal{B}$ is no larger than the sum of the corresponding individual WCET (i.e., $C_{\mathcal{B}} \leq \sum_{\tau_h \in \mathcal{B}} C_h$) by P2. This is equivalent to reducing the execution time of some tasks $\{\tau_h \in \mathcal{B}\}$, such that $C_{\mathcal{B}} = \sum_{\tau_h \in \mathcal{B}} C'_h$, where C'_h denotes the reduced execution time of τ_h. Therefore, the batch execution does not compromise the bounded higher-priority execution of Case F2a. Note that, the existence of τ_j belonging to both $\mathcal{B}$ and $\mathtt{LP}(\tau_k)$ may compromise the bounded higher-priority execution due to additional interference contribution by a lower-priority task τ_j; however, $\mathsf{NPFP}^{\mathcal{B}}$ disallows to execute a batch that belongs to such τ_j.

(Case F2d): We consider a batch (denoted by $\mathcal{B}$) that includes τ_k starts its execution at t $(\geq t_r)$. Different from Case F2c, it is possible for a task whose priority is lower than τ_k to be included in $\mathcal{B}$ due to $\tau_k \in \mathcal{B}$. Recall that $\mathcal{B}$ has the priority of the highest-priority task in $\mathcal{B}$; so, the entire execution of $\mathcal{B}$ has equal or higher priority than τ_k. Since $\mathsf{NPFP}^{\mathcal{B}}$ is work-conserving nonpreemptive scheduling, the execution of $\mathcal{B}$ finishes no later than $t + C_{\mathcal{B}}$, which is no later than $r_k(t) - T_k + R_k = t_r + R_k$ by line 3 of Algorithm 1. We consider two cases: 1) $t_f = t_r + R_k$ and 2) $t_f < t_r + R_k$. In the first case, if we apply F1, the amount of execution of the job of τ_k and jobs of its higher-priority task (by either individual execution or batch execution $\mathcal{B}$) in $[t_r + \Delta_k, t_f)$ is upper-bounded by $R_k - \Delta_k$. Therefore, violation of F2 contradicts (2) in Theorem 1. In the second case, the amount should be strictly less than $C_k + \sum_{\tau_h \in \mathtt{HP}(\tau_k)} \lceil ([t_f - t_r]/T_h) \rceil \cdot C_h$; otherwise, (2) should hold for a value (denoted by $R'_k = t_f - t_r$) that is smaller than R_k.

One may wonder whether the proof for Case F2d correctly considers multiple jobs of $\tau_h \in \mathcal{B} \setminus \{\tau_k\}$ released after t. Since $\mathcal{B}$ starts at t and finishes at t_f, the job of a higher-priority task τ_h released in (t, t_f) will start at t_f or later, which does not belong to $[t_r + \Delta_k, t_f)$, the interval of interest of F2. Instead, the schedulability of the job of a higher-priority task τ_h released in (t, t_f) will be checked by Algorithm 1 when $\tau_k = \tau_h$; if deemed unschedulable, the corresponding $\mathcal{B}$ cannot be scheduled. Therefore, the proof is correct. ■

As shown in line 5 of Algorithm 1, a larger Δ_k implies a higher chance for the algorithm to allow the execution of given $\mathcal{B}$. Therefore, we will use the largest $\{\Delta_k^*\}$ associated with Theorem 1; we already explained how to calculate $\{\Delta_k^*\}$ in Section IV-B. Finally, we present the schedulability analysis of $\mathsf{NPFP}^{\mathcal{B}}$ in the following theorem.

Theorem 3: τ is schedulable by $\mathsf{NPFP}^{\mathcal{B}}$ associated with $\{\Delta_k^*\}$ (i.e., the largest $\{\Delta_k\}$ that makes τ schedulable by Theorem 1), if $\Delta_k^* \geq \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j$ holds for every $\tau_k \in \tau$.

Proof: The theorem holds by Theorems 1 and 2. ■

Run-Time Complexity: At each t, which Algorithm 2 focuses on, we test the $O(\log(|\tau(t)|))$ batch sets by `STOB`, each of which requires $O(|\tau|)$ time-complexity. Therefore, the total run-time complexity is $O(|\tau| \cdot \log(|\tau(t)|)$, which is much lower than $O(|\tau|^2 \cdot |\tau(t)|)$, the complexity of the existing study for scheduling multiple MOT tasks in [3].

VI. $\mathsf{NPFP}^{\mathcal{BI}}$: Exploiting Idling for Batching

Although $\mathsf{NPFP}^{\mathcal{B}}$ efficiently finds and executes a set of active jobs as a batch, it inherently cannot address the situation where there is only one active job at t. Since, the situation cannot be addressed by *any work-conserving scheduling*, we need to develop a run-time idling mechanism that accelerates batch execution to address C1 and C3 in Section I. To this end, we develop $\mathsf{NPFP}^{\mathcal{BI}}$ ($\mathsf{NPFP}^{\mathcal{B}}$ with idling) with given $\{\Delta_k\}$. Based on $\mathsf{NPFP}^{\mathcal{B}}$, $\mathsf{NPFP}^{\mathcal{BI}}$ achieves the same goals: 1) maximizing the batch execution with minimum run-time overhead, while 2) preserving the schedulability guaranteed by Lemma 1.

Our design principles of the run-time idling mechanism of $\mathsf{NPFP}^{\mathcal{BI}}$ are as follows.

1) To prevent idling from compromising F2, any idling cannot be overlapped with any $[t_r + \Delta_i, t_f)$ for any job of τ_i released and finished at t_r and t_f, respectively.
2) Between the execution of a batch set at t and that of the same batch set at t' $(> t)$, the latter cannot help any job's timely execution. Therefore, we restrict time instant candidates at which a batch set starts its execution after idling, to the time instants at which any job (to be executed as a batch) is released.

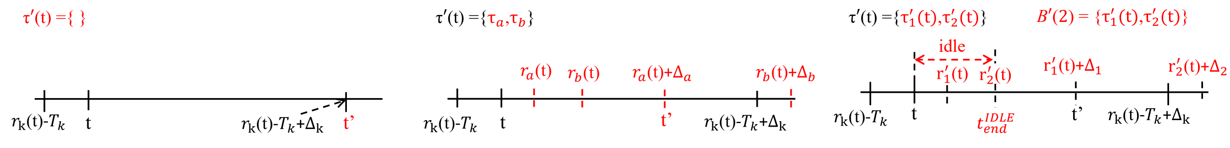


Fig. 5. Example scenario of the idling mechanism of $\mathsf{NPFP}^{\mathcal{BI}}$ in Algorithm 3. (a) Initialization (line 1). (b) Finding candidate tasks (lines 2–5). (c) Finding the largest schedulable batch set (lines 6 and 7).

Algorithm 3 Idling Mechanism of $\mathsf{NPFP}^{\mathcal{BI}}$

1: Set $t' \leftarrow r_k(t) - T_k + \Delta_k$, and $\tau'(t) \leftarrow \emptyset$
2: **for** $\tau_i \in \tau \setminus \{\tau_k\}$ sorted by $r_i(t)$ **do**
3:   **if** $r_i(t) \le t'$ **then** set $\tau'(t) \leftarrow \tau'(t) \cup \{\tau_i\}$, and $t' \leftarrow \min\left(t', r_i(t) + \Delta_i\right)$
4:   **else** Exit the loop.
5: **end for**
6: Let $\tau'_n(t)$ denote the task with the n^{th} earliest next job release time after t among tasks in $\tau'(t)$; $r'_n(t)$ denote the next job release time of $\tau'_n(t)$ after t; and $\mathcal{B}'(n)$ denote $\{\tau'_i(t)\}_{i=1}^{n}$, where $1 \le n \le |\tau'(t)|$.
7: Find the largest batch set $\mathcal{B}'(x) \cup \{\tau_k\}$ for $1 \le x \le |\tau'(t)|$ such that $\mathtt{STOB}\left(r'_x(t), \mathcal{B}'(x) \cup \{\tau_k\}, \mathcal{B}'(x) \cup \{\tau_k\}\right)$ in Algorithm 1 returns schedulable, using the *binary search*; if such $\mathcal{B}'(x)$ exists, set $t_{end}^{\mathsf{IDLE}} \leftarrow r'_x(t)$ where t_{end}^{IDLE} denotes the latest idling time instant, and *return*.
8: Execute the active job of τ_k at t, and *return*.

3) Once we determine to perform batch execution at t after idling, we include all the active jobs to the batch set to be executed, which eases the satisfaction of F1 and F2 in the presence of idling.

Algorithm 3 presents the run-time idling mechanism of $\mathsf{NPFP}^{\mathcal{BI}}$ at t, at which there is only one active job of τ_k while the computing system is ready to work. Lines 1–5 find $\tau'(t) \subset \tau$, a set of candidate tasks to be executed with τ_k as a batch after idling. We aim at calculating $t'(> t)$, which is, the latest time instant at which all of the next released jobs in $\tau'(t)$ and the active job of τ_k can idle without violating F1. t' is determined by the earlier time instant between $r_k(t) - T_k + \Delta_k$ and the earliest one among $r_i(t) + \Delta_i$ for every $\tau_i \in \tau'(t)$. Then, $r_k(t) - T_k + \Delta_k \le t'$ and $r_i(t) + \Delta_i \le t'$ hold for τ_k and every $\tau_i \in \tau'(t)$, respectively, which helps to achieve the first design principle by disallowing any job in the batch set to start its execution after the interval of interest of F2 for the job. In line 6, we define $\tau'_n(t)$ as the task with the n^{th} earliest next job release time after t among tasks in $\tau'(t)$, $r'_n(t)$ as the next job release time of $\tau'_n(t)$ after t, and $\mathcal{B}'(n)$ as $\{\tau'_i(t)\}_{i=1}^{n}$.[4] In line 7, we find the largest x such that the execution of a batch set of $\mathcal{B}'(x) \cup \{\tau_k\}$ does not compromise the schedulability of all the other jobs. To this end, we test $\mathtt{STOB}(r'_x(t), \mathcal{B}'(x) \cup \{\tau_k\}, \mathcal{B}'(x) \cup \{\tau_k\})$ in Algorithm 1, meaning that we check whether a batch set of $\mathcal{B}'(x) \cup \{\tau_k\}$ can start its batch execution at $r'_x(t)$ at which jobs of $\mathcal{B}'(x) \cup \{\tau_k\}$ are the only active jobs; we will explain why it is possible to apply the binary search. If such $\mathcal{B}'(x) \cup \{\tau_k\}$ exists, we reserve that a set of jobs invoked by tasks in $\mathcal{B}'(x) \cup \{\tau_k\}$ will be executed at $r'_x(t)$ by setting $t_{end}^{\mathsf{IDLE}} \leftarrow r'_x(t)$. Otherwise, we execute the single active job at t immediately (in line 8), which is the same as NPFP.

[4]For tasks with the same next job release time, a higher priority implies an earlier next job release time.

Fig. 5 presents an example scenario at the current time instant t, at which an idling decision can be made with τ_k under Algorithm 3. As an initialization, t' and $\tau'(t)$ are set to $r_k(t) - T_k + \Delta_k$ and $\emptyset$, respectively, [line 1 of Algorithm 3, illustrated in Fig. 5(a)]. Then, two tasks τ_a and τ_b will be released at $t < r_a(t)$ and $t < r_b(t)$, and t' and $\tau'(t)$ are updated to $r_a(t) + \Delta_a$ and $\tau'(t) = \{\tau_a, \tau_b\}$, respectively, [lines 2–5 of Algorithm 3, illustrated in Fig. 5(b)]. Finally, $\mathcal{B}'(2) \cup \{\tau_k\}$ is determined as a schedulable batch set according to $\mathtt{STOB}(r'_2(t), \mathcal{B}'(2) \cup \{\tau_k\}, \mathcal{B}'(2) \cup \{\tau_k\})$, and then t_{end}^{IDLE} is set to $r'_2(t)$ [lines 6–7 of Algorithm 3 illustrated in Fig. 5(c)].

We present why it is possible to apply binary search in line 7 of Algorithm 3.

Lemma 3: Recall $r'_n(t)$ and $\mathcal{B}'(n)$ defined in line 6 of Algorithm 3. If $\mathtt{STOB}(r'_{n+1}(t), \mathcal{B}'(n+1) \cup \{\tau_k\}, \mathcal{B}'(n+1) \cup \{\tau_k\})$ in Algorithm 1 returns schedulable, $\mathtt{STOB}(r'_n(t), \mathcal{B}'(n) \cup \{\tau_k\}, \mathcal{B}'(n) \cup \{\tau_k\})$ also returns schedulable.

Proof: If we apply $C_{\mathcal{B}(n)} \le C_{\mathcal{B}(n+1)}$ (from $\mathcal{B}(n) \subset \mathcal{B}(n+1)$ and P3 in Section III) and $r'_n(t) \le r'_{n+1}(t)$ to the opposite conditions in lines 3 and 5 of Algorithm 1, the proof is similar to that of Lemma 2. ■

Including the idling mechanism in Algorithm 3, we present the entire $\mathsf{NPFP}^{\mathcal{BI}}$ scheduling algorithm in Algorithm 4; note that, t_{end}^{IDLE} is set to $-\infty$ when the system starts. In the case of $t < t_{end}^{\mathsf{IDLE}}$ (lines 1 and 2), there should not be any execution, since the idling mechanism determines that all active jobs will be executed at t_{end}^{IDLE}, not at the current time instant t. In the case of $t = t_{end}^{\mathsf{IDLE}}$ (lines 3 and 4), we start to execute all the active jobs as batch execution immediately, which is determined by the idling mechanism. In the case of $t > t_{end}^{\mathsf{IDLE}}$ (lines 5 and 6), we consider two cases. First, if there is only one active job when the computing system is ready to work, we perform Algorithm 3. Second, if there is more than one active job when the computing system is ready to work, we perform Algorithm 2.

Then, we prove that $\mathsf{NP}_{\mathsf{ADAPT}}$ subsumes $\mathsf{NPFP}^{\mathcal{BI}}$ as follows.

Theorem 4: For τ with $\{\Delta_k \ge \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j\}$ that is deemed schedulable by Theorem 1, $\mathsf{NPFP}^{\mathcal{BI}}$ associated with $\{\Delta_k\}$ belongs to $\mathsf{NP}_{\mathsf{ADAPT}}$ associated with $\{\Delta_k\}$.

Proof: Suppose that, a job of τ_k scheduled by $\mathsf{NPFP}^{\mathcal{BI}}$ associated with $\{\Delta_k\}$ (denoted by J_k) is released and finished at t_r and t_f, respectively. We prove that J_k always satisfies F1 and F2 of Definition 1, which proves the theorem.

Algorithm 4 NPFP$^{\mathcal{BI}}$ Scheduling Algorithm

At t, at which a job is finished while there is at least one active job, or at which at least one job is released while the system is idle,

```
if t < t_end^IDLE then
    Any job cannot start its execution.
else if t = t_end^IDLE then
    Execute all the active jobs at t as a batch.
else if t > t_end^IDLE then
    if there is only one active job at t then Perform Algorithm 3.
    else Perform Algorithm 2.
end if
```

First, we check whether F1 is satisfied. Since, we add the run-time idling mechanism to NPFP$^{\mathcal{B}}$, we focus on proving that the idling mechanism does not compromise F1. By the selection of $\tau'(t)$ (in lines 1–5 of Algorithm 3) and the selection of the time instant at which a batch execution starts after idling (in line 7 of Algorithm 3), any batch execution after idling cannot start its execution in $[t_r + \Delta_k, t_f]$. Also, a batch execution after idling can be performed only if line 5 of Algorithm 1 guarantees that the execution finishes before $t_r + \Delta_k$. This proves the satisfaction of F1.

Second, we check whether F2 is satisfied. Since, the idling mechanism complies with F1, we can check F2 when a batch execution starts. This is achieved by calling Algorithm 1 by line 7 of Algorithm 3, which corresponds to line 3 of Algorithm 2 for NPFP$^{\mathcal{B}}$. Therefore, the remaining proof is similar to that for NPFP$^{\mathcal{B}}$ in Theorem 2. ∎

Finally, we present the schedulability analysis of NPFP$^{\mathcal{BI}}$ in the following theorem.

Theorem 5: τ is schedulable by NPFP$^{\mathcal{BI}}$ associated with $\{\Delta_k^*\}$ (i.e., the largest $\{\Delta_k\}$ that makes τ schedulable by Theorem 1), if $\Delta_k^* \geq \max_{\tau_j \in \mathtt{LP}(\tau_k)} C_j$ holds for every $\tau_k \in \tau$.

Proof: The theorem holds by Theorems 1 and 4. ∎

Run Time-Complexity: At each t, which Algorithm 3 focuses on lines 1–5 check at most $|\tau|$ tasks, and lines 6–8 test the $O(\log(|\tau(t)|))$ batch sets by STOB, each of which requires $O(|\tau|)$ time complexity; hence, the total run-time complexity of Algorithm 3 is $O(|\tau| \cdot \log(|\tau(t)|)$. By the run-time complexity of Algorithms 2 and 3, that of Algorithm 4 is also $O(|\tau| \cdot \log(|\tau(t)|)$, which is much lower than $O(|\tau|^2 \cdot |\tau(t)|)$ for the existing study [3].

VII. Evaluation

A. Experiment Setup

We consider four different computing systems. The first one is equipped with Intel Xeon Silver 4215R CPUs @ 3.20 GHz, 251.5 GB RAM, and a Tesla V100 GPU. We also consider three GPU-enabled embedded boards: 1) NVIDIA Jetson TX2; 2) Xavier; and 3) Orin. The MOT execution pipeline and scheduler run on Python and Pytorch, and the model precision is set to FP16; on the same experiment setting in Fig. 1, we observed nearly the same tracking accuracy with FP32 owing to the mixed precision training. As the object detector, we consider the YOLO series [13], [20] trained with the COCO Dataset [24]. We use SORT [6] as the object tracker of the two-stage methods. The performance was evaluated using the Waymo Open Dataset [25], the autonomous driving data set collected by autonomous driving cars. For the evaluation, we use the measured WCETs in Fig. 8.

B. Experiment Results

In this section, we demonstrate the effectiveness of the proposed run-time batching and idling mechanisms in improving the tracking accuracy by comparing the following three approaches that operate on the architecture of Batch-MOT. Note that, we apply the rate monotonic (RM) [26] to FP, for all the approaches in this section.

1) NPFP in Section IV-A, in which all the MOT tasks perform individual execution with down-scaled images.
2) NPFP$^{\mathcal{B}}$ in Section V, in which down-scaled and full-size images are processed for the individual and batch execution, respectively.
3) NPFP$^{\mathcal{BI}}$ in Section VI, in which down-scaled and full-size images are processed for the individual and batch execution, respectively.

We also compare our approaches with the only existing study that addresses G1 and G2 for multiple MOT tasks as follows.

1) RT-MOT, a flexible MOT execution scheduling framework on the architecture proposed in [3] with the YOLO series and DeepSORT [5] as its detector and tracker.

For all the target approaches, we provide a run-time option called the individual full-size execution policy (IFP). Under IFP, each target approach performs a full-size execution at t only if: i) an MOT task τ_i is active alone at t; ii) the full-size execution (even with its WCET) of the only active task at t can be completed by $t_{\mathtt{next}}$ (the earliest future release of any task later than t), i.e., $t + C_i^{\mathtt{full}} \leq t_{\mathtt{next}}$, where $C_i^{\mathtt{full}}$ is the WCET for the full-size execution of the task; and iii) if the target approach is NPFP$^{\mathcal{BI}}$, the system is not idle at t under NPFP$^{\mathcal{BI}}$, i.e., $t < t_{end}^{\mathsf{IDLE}}$ in Algorithm 3. Conditions i) and ii) ensure IFP improves accuracy through the full-size execution without compromising timing guarantees of the other tasks. Condition iii) ensures IFP can be incorporated into NPFP$^{\mathcal{BI}}$ without conflicting with its idling mechanism. Note that, RT-MOT inherently employs IFP.

While the existing DNN-based MOD techniques (e.g., [2], [10], and [11]) could be considered for comparison, adapting them for the MOT systems would require new contributions. For example, extending DNN-SAM [2] would need alignment in tracking algorithm, ROI identification, and other features of RT-MOT for fairness. Although DNN-SAM is optimized for accuracy within the ROI, it requires two separate DNN inferences: one for the ROI and another for outside areas, hindering high overall accuracy. Executing two DNN inferences not only doubles the computational tasks of pre/postprocessing [as shown in i) and iii) of Fig. 1], but also limits the benefits of increased GPU utilization from batch execution.

Since, the approaches share the same offline schedulability test in Lemma 1, we compare their tracking accuracy of the task sets whose schedulability is guaranteed by the test. We use multiple object tracking accuracy (MOTA) [27], a primary metric to evaluate the tracking accuracy; tracking accuracy

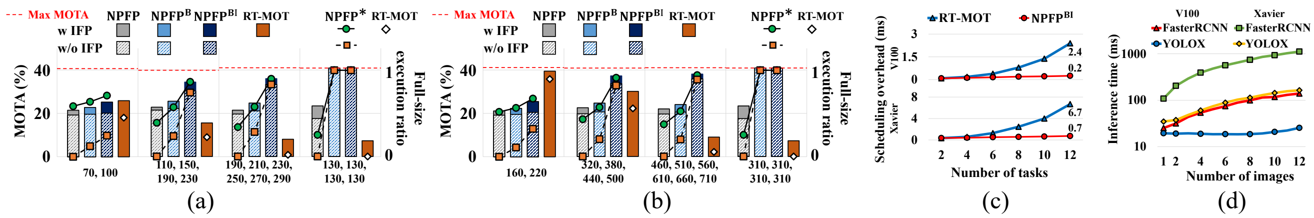


Fig. 6. Comparison of different approaches on YOLOX (a), (b), and (c), and different DNN models (d).

under MOTA is derived by counting miss detection, false detection, and miss tracking; we obtained similar experimental results using IDF-1 [27], another widely recognized metric for the tracking accuracy. In addition, to evaluate the effectiveness of resource utilization of each approach, we measure the ratio of the number of full-size image executions to the total number of the MOT executions (referred to as the full-size execution ratio).

Fig. 6(a) and (b) compare the tracking accuracy and full-size execution ratio of the four approaches using YOLOX on the Tesla V100 and Jetson Xavier. Similar results were observed for the Jetson TX2 and Orin (also with YOLOv5 [20]). We consider four sets of MOT tasks [periods shown on the x-axis in Fig. 6(a) and (b)] that pass the test in Lemma 1, but schedulability is not guaranteed when all the tasks use full-size input images. Note that, the two computing systems provide different WCETs, resulting in different task periods in each set. The bar and line in each graph represent the average MOTA score and full-size execution ratio, respectively. The red dotted line indicates the maximum achievable tracking accuracy.

As shown in Fig. 6(a) and (b), a higher full-size execution ratio leads to higher accuracy. The accuracy of RT-MOT shows a significant decrease when the full-size execution ratio is low as observed in the third and fourth task sets of Fig. 6(a) and (b). This decline is attributed to the unique approach of RT-MOT, where it detects objects only in a partial region (i.e., the region of interest) of the input image when the full-size execution cannot be performed. In contrast, $\text{NPFP}^{\mathcal{B}}$ and $\text{NPFP}^{\mathcal{BI}}$ consider the down-scaled entire region, making them more resilient to a low full-size execution ratio. As the number of tasks increases, the accuracy of RT-MOT dramatically decreases for both the computing systems. However, $\text{NPFP}^{\mathcal{B}}$ and $\text{NPFP}^{\mathcal{BI}}$ maintain high accuracy by securing the chance of batch execution. In the case of a set of tasks with equal periods (e.g., the fourth task set), $\text{NPFP}^{\mathcal{B}}$ and $\text{NPFP}^{\mathcal{BI}}$ achieve maximum accuracy owing to the high chance of batch execution. The IFP approach contributes significantly to improving accuracy in both the computing systems.

Fig. 6(c) presents the run-time overhead of the scheduling algorithm for $\text{NPFP}^{\mathcal{BI}}$ and RT-MOT. As discussed in Section VI, the run-time complexity of $\text{NPFP}^{\mathcal{BI}}$ is $O(|\tau| \cdot \log(|\tau(t)|)$, which is much lower than $O(|\tau|^2 \cdot |\tau(t)|)$, the complexity of RT-MOT. As the number of MOT tasks increases, the difference between the run-time scheduling overhead of $\text{NPFP}^{\mathcal{BI}}$ and RT-MOT becomes larger. For example, the run-time scheduling overhead of RT-MOT is about 12 times (2.4/0.2) and 9.6 times (6.7/0.7) larger than that of $\text{NPFP}^{\mathcal{BI}}$ for a set of 12 MOT tasks on the two considered systems, respectively.

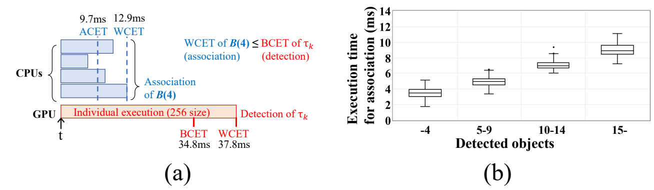


Fig. 7. CPU/GPU parallel execution example. (a) Varying WCET with the different number of objects on Jetson Orin. (b) Varying association WCET.

Fig. 6(d) shows the variation in average DNN inference time (excluding pre/post processing) based on the number of input images for batch execution, using different DNN models and computing systems. YOLOX demonstrates GPU resource saturation with more than the ten input images on Tesla V100 and two on Jetson Xavier, highlighting the effectiveness of batch execution in reducing inference time; similar trends were observed for YOLOv5. In contrast, Faster-RCNN [21] saturates with a single input image due to its two-stage design, which splits computation into region proposal and classification, resulting in higher serialization during classification as noted in [4].

VIII. Discussion

CPU/GPU Parallelism: Contrary to Batch-MOT's assumption, consider CPU/GPU parallel execution where, at time t, four associations of $\mathcal{B}(4)$ are executed in parallel on multiple CPUs, while the another task τ_k is performed individually on the GPU as shown in Fig. 7(a). If the WCET of $\mathcal{B}(4)$'s association (e.g., 12.9 ms) is less than the best case execution time (BCET) of τ_k (e.g., 34.8 ms), then τ_k's association can be executed nonpreemptively, consistent with Batch-MOT's assumption. Our experiments confirmed that this condition always holds. Then, we conducted the accuracy evaluation, including CPU/GPU parallel execution on Jetson Orin without modifying the Batch-MOT's offline tests. The experiments demonstrated that the CPU/GPU parallel execution did not incur any deadline misses and resulted in a marginal accuracy improvement (up to 0.81%) compared to the case without CPU/GPU parallel execution (see Figure S.4(a) in supplement). The reason for the marginal improvement is that, as shown in Figure S.4(a), the average case execution time (ACET) of the association (e.g., 9.7 ms) is much smaller than that of detection, so the reduction in response time achieved by CPU/GPU parallel execution is minimal.

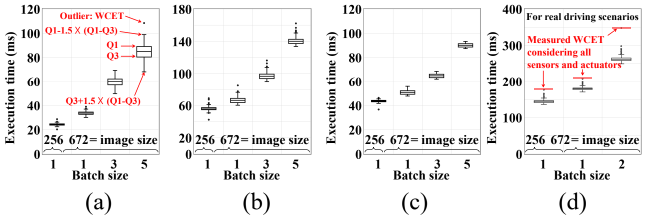


Fig. 8. Execution time measurements on Waymo Dataset (a)–(c) and our real-world driving scenarios (d).

Flexible WCET: Association compares objects detected in the t-th frame with those in the $(t-1)$th frame, associating the most similar pairs. Therefore, the WCET of association in the t-th frame depends strictly on the number of detected objects as shown in Fig. 7(b). If Batch-MOT splits an MOT task into detection and association subtasks and allows scheduling decisions between them, the WCET of the association subtask (measured offline based on the number of objects) can be dynamically determined by the number of objects detected in the detection subtask. This approach would require additional alters to the Batch-MOT's current schedulability tests.

WCET Measurement: Fig. 8 shows the execution time for the down-scaled (256×256) images and batch execution of full-size (672×672) images for up to five MOT tasks using YOLOX on the four computing systems evaluated in Section VII. Measurements are with 1000 iterations to obtain WCET [e.g., Outlier: WCET in Fig. 8(a)]; more details are in Figure S.2 and Table S.2 in the supplement. The Waymo Dataset was used for Tesla V100, Jetson Xavier, and Orin, while real driving scenario videos were used for Jetson TX2. Fig. 8(d) considers all the communication overheads, including sensors (e.g., camera and LiDAR) and actuators. Note that, Batch-MOT does not predict execution time at run-time but uses offline WCET. Recent studies on offline WCET of DNN execution [2], [3], [4], [11], [18] are widely accepted. It is generally reasonable to assume an upper bound with high confidence through intensive measurement plus a safety margin.

IX. Conclusion

In this article, we proposed a novel system design, Batch-MOT, that enables batch execution of multiple MOT tasks to maximize the tracking accuracy while providing timing guarantees. Using a new scheduling framework, $\mathsf{NP}_{\mathsf{ADAPT}}$, which allows run-time execution deviations with timing guarantees, we developed a run-time batching mechanism, $\mathsf{NPFP}^{\mathcal{B}}$, and a run-time idling mechanism, $\mathsf{NPFP}^{\mathcal{BI}}$. These mechanisms efficiently find and execute MOT tasks as a batch without compromising timely execution. Experiments demonstrated that Batch-MOT improves the tracking accuracy over the state-of-the-art real-time MOT systems while ensuring timing guarantees.

References

[1] M. Yang et al., "Re-thinking CNN frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge," in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, 2019, pp. 305–317.

[2] W. Kang et al., "DNN-SAM: Split-and-merge DNN execution for real-time object detection," in *Proc. 28th IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, 2022, pp. 160–172.

[3] D. Kang et al., "RT-MOT: Confidence-aware real-time scheduling framework for multi-object tracking tasks," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2022, pp. 318–330.

[4] S. Liu et al., "Self-cueing real-time attention scheduling in criticality-aware visual machine perception," in *Proc. IEEE Real Time Technol. Appl. Symp. (RTAS)*, 2022, pp. 173–186.

[5] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, 2017, pp. 3645–3649.

[6] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, 2016, pp. 3464–3468.

[7] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, "FairMOT: On the fairness of detection and re-identification in multiple object tracking," *Int. J. Comput. Vis.*, vol. 129, no. 11, pp. 3069–3087, 2021.

[8] P. Chu, J. Wang, Q. You, H. Ling, and Z. Liu, "TransMOT: Spatial-temporal graph transformer for multiple object tracking," 2021, *arXiv:2104.00194*.

[9] J. Pang et al., "Quasi-dense similarity learning for multiple object tracking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 164–173.

[10] H. Zhou, S. Bateni, and C. Liu, "S^{3}DNN: Supervised streaming and scheduling for GPU-accelerated real-time DNN workloads," in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, 2018, pp. 190–201.

[11] Y. Xiang and H. Kim, "Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2019, pp. 392–405.

[12] "NVIDIA Orin developer kit." Accessed: Mar. 27, 2023. [Online]. Available: https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems/jetson-orin/

[13] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," 2021, *arXiv:2107.08430*.

[14] "NVIDIA xavier developer kit." Accessed: Jul. 9, 2018. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier

[15] "Supplement." Accessed: Feb. 8, 2024. [Online]. Available: https://www.bit.ly/24EMSOFT-Batch-MOT-supplement

[16] A. Soyyigit, S. Yao, and H. Yun, "Anytime-Lidar: Deadline-aware 3D object detection," in *Proc. IEEE Int. Conf. Embed. Real-Time Comput. Syst. Appl. (RTCSA)*, 2022, pp. 31–40.

[17] S. Heo, S. Jeong, and H. Kim, "RTScale: Sensitivity-aware adaptive image scaling for real-time object detection," in *Proc. Leibniz Int. Proc. Inform. (LIPIcs)*, vol. 231, 2022, pp. 1–22.

[18] S. Lee and S. Nirjon, "SubFlow: A dynamic induced-subgraph strategy toward real-time DNN inference and training," in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, 2020, pp. 15–29.

[19] S. Liu et al., "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2020, pp. 319–332.

[20] "YOLOv5." Accessed: Nov. 23, 2022. [Online]. Available: [Online]. Available: https://github.com/ultralytics/yolov5

[21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.

[22] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," INRIA, Paris, France, Rep. RR-2966, 1996. [Online]. Available: https://who.rocq.inria.fr/Laurent.George/#Publication

[23] G. Yao, G. Buttazzo, and M. Bertogna, "Feasibility analysis under fixed priority scheduling with fixed preemption points," in *Proc. IEEE Int. Conf. Embed. Real-Time Comput. Syst. Appl. (RTCSA)*, 2010, pp. 71–80.

[24] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. 13th Eur. Conf. Comput. Vis. (ECCV)*, 2014, pp. 740–755.

[25] P. Sun et al., "Scalability in perception for autonomous driving: Waymo open dataset," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 2446–2454.

[26] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 1989, pp. 166–171.

[27] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in *Proc. Eur. Conf. Comput. Vis. Workshops (ECCVW)*, 2016, pp. 17–35.