Real-time Task Scheduling for Digital Twin Edge Network

Cheonyong Kim[1], Mahdi Chehimi[2], Minchae Jung[1], and Walid Saad[2,3]
[1]Department of Electronics and Information Engineering, Sejong University, Seoul, Republic of Korea
Emails: {cykim0807, mcjung}@sejong.ac.kr
[2]Wireless@VT, Bradley Department of Electrical and Computer Engineering, Virginia Tech, VA, USA
Emails: {mahdic, walids}@vt.edu
[3]Cyber Security Systems and Applied AI Research Center, Lebanese American University, Lebanon

***Abstract*—The deployment of digital twins (DTs) at the edge of a wireless network can facilitate low-latency and high-throughput DT autonomous and real-time Internet of everything (IoE) applications. In such DT edge networks (DTENs), each DT has two types of real-time tasks that require timely processing: DT update tasks and DT inference tasks. However, the joint scheduling of these two types of tasks has been overlooked in prior works. In this paper, the first joint real-time scheduling scheme for DT update and inference tasks in a DTEN is proposed. Moreover, a novel performance metric called freshness is introduced to capture the effectiveness and synchronization performance of scheduling. Also, a new scheduling scheme is proposed to efficiently solve a freshness maximization problem for DTENs. Simulation results show that the performance of the proposed scheme is within 4% of the upper bound for DTENs with 20 physical objects, and within 12% of the upper bound in worst cases for DTENs with more than 30 physical objects. The results also show that the proposed approach reduces the maximum de-synchronization time by 63% compared to existing real-time scheduling algorithms.**

***Index Terms*—Digital twin, edge network, IoE, real-time task scheduling.**

I. Introduction

Digital twins (DTs) are a transformative technology that teleports physical objects and their operations to the virtual world to optimize operational efficiency. DTs go beyond a simple replica of the physical reality, as they enable monitoring and controlling the performance of physical twins (PTs) through instantaneous real-time feedback [1]. Recently, DTs have gained significant interest from diverse industries as well as academic researchers due to their potential to replace conventional testing and simulation mechanisms that are resource-intensive and time-consuming [2].

Despite their great potential, DT implementation faces significant challenges. One key challenge is ensuring that DTs accurately represent the dynamic behavior of their PTs in real-time. This requires meeting strict requirements in terms of computation and communication latency. For instance, using a centralized cloud computing infrastructure for DT implementation may result in high communication delays. To address these issues and improve the performance and efficiency of DTs, a decentralized architecture using edge servers (ESs) can be used to deploy the DTs. Such an architecture forms a *DT edge network (DTEN)* and enables DT services with low computation and communication delay [3]. By integrating DTs with an intelligent edge backbone in a DTEN, a high-quality representation of dynamic DTs can be ensured, facilitating the deployment of DTs in non-stationary real-world scenarios for autonomous Internet of everything (IoE) applications.

In general, a DT application entails two types of DT tasks that must be concurrently fulfilled by the ES. This includes the *update task* for updating the DT model using the collected data from its PT, and the *inference task* for obtaining an inference or making predictions from a DT without interrupting its PT. Moreover, an update task should be processed as soon as possible for synchronizing the DT with its PT whereas an inference task has a deadline for real-time response. Considering both types of DT tasks and scheduling their processing in real-time in a DTEN is a challenging problem that was overlooked in most prior works [3]–[6].

Indeed, several prior works [3]–[6] studied DT implementations over DTENs while considering the two types of DT tasks, separately. For instance, the work in [4] investigated the service congestion issue caused by stochastic DT service requests and proposed a long-term congestion control scheme. However, this prior work [4] takes into account only the inference task without considering the update task. On the other hand, the works in [3], [5] and [6] considered the update task without considering the inference task. For instance, the work in [3] investigated the update task of DTs in DTENs to ensure reliability and overcome non-stationary effects. The authors in [5] formulated a DT-edge association problem to reduce the system latency and enhance the user utility. The work in [6] designed a blockchain-empowered federated learning framework for building a global model of distributed DTs. However, no prior work considered the joint scheduling of both update and inference tasks in DTENs, which remains an open problem. Furthermore, the works in [4] and [5] limit the number of DTs that an ES can maintain considering one type of DT task, which results in difficulty in accommodating the other one. Since an improperly designed

This work was supported in part by Institute of Information & Communications Technology Planning & Evaluation (IITP) under the metaverse support program to nurture the best talents (IITP-2023-RS-2023-00254529) grant funded by the Korea government(MSIT) and by the U.S. National Science Foundation under Grant CNS-1814477.

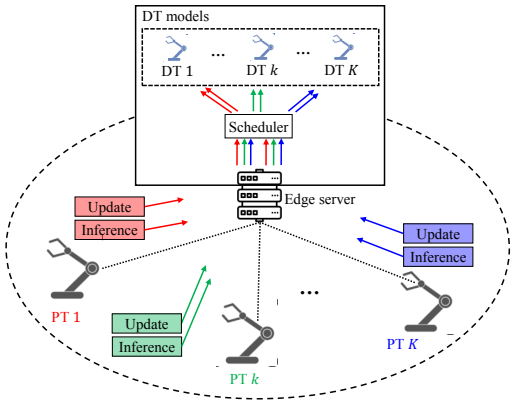

Fig. 1. Illustrative system model of a DTEN consisting of multiple PTs and an ES.

DT task scheduler may cause delayed DT synchronizations or violations of deadlines for inference requests, *there is a need to carefully design real-time scheduling schemes that jointly consider update and inference tasks in DTENs.*

The main contribution of this work is a novel approach for jointly scheduling real-time DT update and inference tasks in DTENs. Toward achieving this overarching goal, we make the following key contributions:

- We perform the first thorough analysis of a DTEN environment that captures both update and inference DT tasks, which is fundamental for areas like smart factories, autonomous vehicles, and personalized health care. The considered scenario jointly encompasses PT-generated real-time data for updating DTs, along with users' requests for real-time responses for predictions and inferences from the updated DTs in the DTEN.
- We propose a novel performance metric named *freshness* for estimating real-time DT task scheduling performance. We also formulate a freshness maximization problem to enhance DT effectiveness and synchronization performance. Moreover, we introduce an efficient scheduling scheme to solve the proposed problem by transforming it into an iterative integer programming problem.
- Simulation results show that the proposed scheme maximizes the freshness, and achieves a near-optimal performance, i.e., within 4% of the upper limit with 20 PTs, and within 12% of the upper limit in the worst cases of > 30 PTs. In addition, the proposed scheme reduces the maximum de-synchronization time by up to 63% compared to existing real-time scheduling algorithms.

II. System Model

We consider a DTEN that incorporates an ES connected to a set $\mathcal{K}$ of K PTs each of which is the requester of update and inference tasks, as shown in Fig. 1. The ES maintains K DTs each of which is implemented using a deep neural network (DNN) model. Since a DT has to reflect the current status of its PT, we adopt a continual learning approach [3] instead of using a trained model. Therefore, a PT periodically generates update tasks including the data collected from sensors. On the other hand, a DT can be used for testing or simulating the reaction of the PT without disturbing the PT's operation. With this purpose, a PT also generates inference tasks for obtaining insights from the updated DT. The considered system undergoes sequential episodes each of which consists of several rounds. This scenario models several practical use cases and applications such as manufacturing [7] and healthcare [8]. For example, in a smart factory, a specific product is manufactured within a given round and the system resources are episodically reconfigured when beginning to manufacture another type of product.

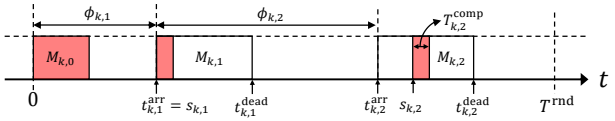

Fig. 2. An example of the arrivals of DT tasks within a round.

A. DT task model

We assume that the arrivals of the DT tasks at the ES are fixed within a round but change over episodes. During a round, each PT generates one update task and multiple inference tasks. A task for DT k is represented by $M_{k,y} = \left(t^{\text{arr}}_{k,y}, b_{k,y}, t^{\text{dead}}_{k,y}\right)$ where y is the task index, $t^{\text{arr}}_{k,y}$ is the arrival time, $b_{k,y}$ is the data size, and $t^{\text{dead}}_{k,y}$ is the deadline. Update tasks are differentiated from inference tasks by using the task index y. For DT k, $M_{k,0}$ denotes the update task whereas inference tasks are represented by $M_{k,y}$, $\forall y \in \mathcal{Y}_k = \{1, ..., Y_k\}$ where Y_k is the total number of inference tasks. We assume that inference tasks follow the sporadic task model and thus, they have a minimum inter-arrival time between two consecutive tasks [9]. The minimum and maximum inter-arrival times are, respectively, $\phi^{\min}$ and $\phi^{\max}$. The feasible range $\phi_{k,y}$ of the interval between $M_{k,y-1}$ and $M_{k,y}$ is bounded, i.e., $\phi^{\min} \leq \phi_{k,y} \leq \phi^{\max}$. Given the duration of a round T^{rnd}, the maximum number of inference tasks can be obtained as $Y^{\max} = \lfloor T^{\text{rnd}}/\phi^{\min} \rfloor$. Since $M_{k,y}$ must be processed before the arrival of $M_{k,y+1}$, The deadline of $M_{k,y}$ is given by $t^{\text{dead}}_{k,y} = t^{\text{arr}}_{k,y} + \phi^{\min}$, $\forall y \in \mathcal{Y}_k$. An inference task contains one-time data whose size is δ and, thus, we have $b_{k,y} = \delta, \forall y \in \mathcal{Y}_k$. On the other hand, by separately allocating communication resources to each PT, all update tasks arrive at the beginning of a round (i.e., $t^{\text{arr}}_{k,0} = 0$). In addition, all update tasks must be processed within a round such that $t^{\text{dead}}_{k,0} = T^{\text{rnd}}$. An update task contains data from several samples, such that $b_{k,0} = \alpha\delta$ where α is the number of samples captured during a round. Fig. 2 shows an example of the arrivals of tasks for DT k during a round.

B. Computing model

We assume that the ES processes DT tasks sequentially according to the scheduler output without idling the processor[1]. In Fig. 2, $s_{k,y}$ is the time that the computation of $M_{k,y}$ begins

[1]This paper focuses on the fundamental characteristics of real-time DT task scheduling in a basic system model. Practical computing issues such as multi-processor and preemption will be covered in our future work.

and $T_{k,y}^{\mathrm{comp}}$ is the delay for processing $M_{k,y}$. $T_{k,y}^{\mathrm{comp}}$ is obtained as $T_{k,y}^{\mathrm{comp}} = \frac{b_{k,y}\omega}{f^{\mathrm{edge}}}$ where ω is the complexity of a DT task in cycles/bit and f^{edge} is the computational capacity of the ES in cycles/s. Considering that PTs are involved in the same DT application where $b_{k,y} = \delta$ and $b_{k,0} = \alpha\delta$ $\forall k \in \mathcal{K}, \forall y \in \mathcal{Y}_k$, we define $T_{\mathrm{upd}}^{\mathrm{comp}}$ as the computation delay of an update task and $T_{\mathrm{inf}}^{\mathrm{comp}}$ as the computation delay of an inference task.

C. Scheduling model

The scheduler prioritizes the DT tasks by adjusting $s_{k,y}$ within the feasible range between $t_{k,y}^{\mathrm{arr}}$ and $t_{k,y}^{\mathrm{dead}} - T_{k,y}^{\mathrm{comp}}$. Given a set of all DT tasks, the total number of DT tasks Q will be $Q = K + \sum_{k=1}^{K} Y_k$. Therefore, a pair of k and y can be replaced by $q \in \{1, ..., Q\}$. Let $\varphi(q)$ be the priority of M_q. Then, a scheduler output $\boldsymbol{\varphi} \in \mathbb{N}^Q$ can be seen as a permutation of the DT task set and expressed as follows:

$$\boldsymbol{\varphi} = \begin{pmatrix} 1 & \cdots & Q \\ \varphi(1) & \cdots & \varphi(Q) \end{pmatrix}, \tag{1}$$

where the smaller value of $\varphi(q)$ indicates the higher priority. We also define $\psi(p)$ to be the index of a certain task whose priority is p such that $\psi(p) = q$ if $\varphi(q) = p$.

Now we assess the capacity of the ES by estimating whether the DT tasks can be processed within their deadlines without any violations. In a round, there will be K update tasks and the maximum number of inference tasks is $KY^{\max}$. Since the deadline for update tasks is T^{rnd}, the condition by which the update tasks can be scheduled is derived as follows:

$$KT_{\mathrm{upd}}^{\mathrm{comp}} + KY^{\max}T_{\mathrm{inf}}^{\mathrm{comp}} \leq T^{\mathrm{rnd}}. \tag{2}$$

In the case of inference tasks, we consider the worst-case scenario in which the ES has started processing an update task at t and K inference tasks arrived at $t+\epsilon$ simultaneously. If $\epsilon \approx 0$, the ES should be able to process one update task and K inference tasks within $\phi^{\min}$. Therefore, the condition by which the inference tasks can be scheduled is obtained by

$$T_{\mathrm{upd}}^{\mathrm{comp}} + KT_{\mathrm{inf}}^{\mathrm{comp}} \leq \phi^{\min}. \tag{3}$$

These conditions can be used to configure the system parameters of a DTEN. For instance, the operator may reduce the number of samples α to satisfy a shorter deadline of inference tasks or extend the relative deadline of inference tasks to accommodate more PTs in the system. We assume that these conditions are satisfied and focus on the DT-specific characteristics of real-time DT task scheduling.

Given the system model, our goal is to develop a real-time task scheduling scheme that is desirable in DT applications. To this end, we introduce a novel performance metric by scrutinizing the properties of DT task scheduling which differs from general real-time task scheduling.

III. Real-time DT Task Scheduling

In this section, we analyze the characteristics of DT task scheduling and propose a new performance metric for estimating the quality of a DT task scheduler. In addition, we propose a DT task scheduling scheme by formulating an optimization problem to maximize the performance metric.

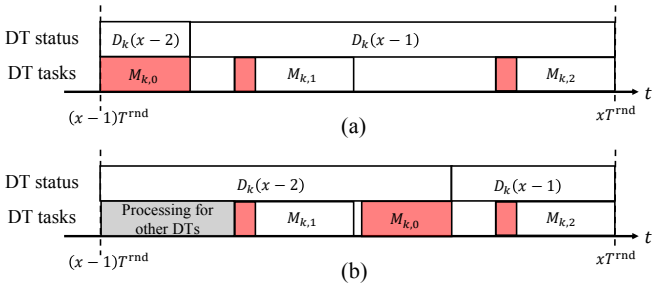

Fig. 3. Examples of the freshness; (a) the case of maximum freshness ($F(k;\boldsymbol{\varphi}) = 2 = Y_k$) and (b) the case of compromised freshness ($F(k;\boldsymbol{\varphi}) = 1 < Y_k$)

A. Characteristics of DT task scheduling

From the perspective of real-time task scheduling, an update task should be processed quickly to synchronize the PT and its DT whereas an inference task must meet a desired deadline to ensure a real-time response. With this understanding, we can consider two primitive approaches; update-first and inference-first schedulers. The update-first approach may violate the deadline of inference tasks whereas the inference-first scheduler will lead to a high and possibly unacceptable desynchronization time, which is defined as the duration between generation and completion of an update task. Therefore, a real-time DT task scheduling scheme must be prudently designed by jointly considering update and inference DT tasks.

Instead of priority, we focus on the processing of update tasks and their impact on the status of the DT models. Fig. 3 shows two examples of different scheduler outputs. Let $D_k(x)$ be DT k updated using the data from round x. Obviously, the update task arriving at the beginning of round x contains the data from round $x-1$. Thus, the status of DT k changes from $D_k(x-2)$ to $D_k(x-1)$ after processing $M_{k,0}$. Meanwhile, inference tasks arriving during round x should be processed using the *latest* DT model (i.e., $D_k(x-1)$) as shown in Fig. 3 (a). However, some inference tasks may not be processed *after* the update task has been processed for processing other tasks, as shown in Fig. 3 (b). Therefore, given a limited computing power of the ES, a scheduler should maximize the number of inference tasks that are processed using the latest DT models.

Based on the above characteristics, we introduce the concept of *freshness* as a novel performance metric for real-time DT task scheduling. The freshness of a DT can be quantitatively defined as *the number of inference tasks that are processed after the update task has been processed*[2]. Given a scheduler output $\boldsymbol{\varphi}$, the *freshness* of DT k, $F(k;\boldsymbol{\varphi})$, can be obtained by

$$F(k;\boldsymbol{\varphi}) = \sum_{i=1}^{Y_k} \max\left(\frac{\varphi(k,i) - \varphi(k,0)}{|\varphi(k,i) - \varphi(k,0)|}, 0\right). \tag{4}$$

Using (4), next, we pose the DT task scheduling problem as a freshness maximization problem.

[2]*Freshness* may be defined using the age of information concept [10] (e.g., the elapsed time after a DT has been updated). On the other hand, our definition focuses on the relationship between update/inference tasks and the status of DT when inference tasks are processed.

B. Freshness maximization problem

The freshness maximization problem can be formulated as follows:

$$\mathcal{P}: \max_{\boldsymbol{\varphi}} \Theta(\boldsymbol{\varphi}) = \sum_{k=1}^{K} F(k;\boldsymbol{\varphi}) \tag{5}$$

$$\text{s.t. } t_{k,y}^{\text{arr}} \le s_{k,y} \le t_{k,y}^{\text{dead}} - T_{k,y}^{\text{comp}}, \tag{5a}$$

$$s_{k,y} + T_{k,y}^{\text{comp}} \le s_{k',y'}, \text{if } \varphi(k,y) < \varphi(k',y'), \tag{5b}$$

where (5a) is the feasible range of $s_{k,y}$ and (5b) denotes that the task whose priority is higher must be processed earlier. In (5), $\mathcal{P}$ is a combinatorial problem. Since the feasible solution set cannot be directly obtained because a) inference tasks must meet different deadlines according to (5a), and b) the objective function $\Theta(\boldsymbol{\varphi})$ is not linear, then $\mathcal{P}$ cannot be directly solved. Although the problem in (5) can be solved by an exhaustive search method, this approach is computationally infeasible when the number of PTs is large.

C. Transformation into iterative integer programming

To solve $\mathcal{P}$ in polynomial time, we exploit two observations on DT task scheduling. First, for maximum freshness, update tasks should be processed as soon as possible unless they breach inference task deadlines. In other words, the computation of inference tasks should be delayed as close to their deadlines as possible to make room for update tasks. Second, update tasks can be processed successively when there are no inference tasks to be processed, while some inference tasks should be processed without intervention if they have similar deadlines. As a result, the scheduler performs iterative processing of update tasks followed by inference tasks.

From these observations, we transform $\mathcal{P}$ into an iterative integer programming problem. Specifically, we first group the inference tasks according to their deadlines. We refer to an inference task group as an atomic set (AS), as their processing is conducted without intervention. Then, we formulate an integer programming problem to schedule a suitable set of update tasks between two consecutive ASs, with the aim of maximizing the total freshness.

The process of building ASs consists of two phases. First, given the arrivals of inference tasks, as shown in Fig. 4 (a), the scheduler aligns the computations of all inference tasks to their deadlines resulting in $s_{k,y} = t_{k,y}^{\text{dead}} - T_{\text{inf}}^{\text{comp}}$. Then, the initial priorities of the inference tasks are assigned in ascending order of $t_{k,y}^{\text{dead}}$ satisfying $\varphi(k,y) < \varphi(k',y')$, if $t_{k,y}^{\text{dead}} < t_{k',y'}^{\text{dead}}$, as shown in Fig. 4 (b). In the second phase, the computations of all inference tasks are rearranged to be aligned with $T_{\text{upd}}^{\text{comp}}$. Let $\mathcal{S}^{(i)}$ be the AS of iteration i. For $\mathcal{S}^{(i)}$, $t_{\mathcal{S}^{(i)}}^{\text{start}}$ and $t_{\mathcal{S}^{(i)}}^{\text{end}}$ are defined as the times when the processing of $\mathcal{S}^{(i)}$ begins and terminates, respectively. We also define $z^{(i)}$ and $\boldsymbol{r}^{(i)} = [r_1^{(i)}, ..., r_K^{(i)}]^{\mathrm{T}}$ as, respectively, the maximum number of update tasks that can be scheduled between $t_{\mathcal{S}^{(i-1)}}^{\text{end}}$ and $t_{\mathcal{S}^{(i)}}^{\text{start}}$, and the indication vector for the existence of inference tasks in $\mathcal{S}^{(i)}$. The rearrangement is conducted in order of $M_{\psi(1)}, ..., M_{\psi(Q-K)}$. From $i = 0$ ($t_{\mathcal{S}^{(0)}}^{\text{end}} = 0$), the scheduler compares $t_{\mathcal{S}^{(i)}}^{\text{end}}$ and $s_{\psi(p)}$. If $\lfloor (s_{\psi(p)} - t_{\mathcal{S}^{(i)}}^{\text{end}})/T_{\text{upd}}^{\text{comp}} \rfloor > 0$, the scheduler creates $\mathcal{S}^{(i+1)}$ and inserts $M_{\psi(p)}$ into $\mathcal{S}^{(i+1)}$ (i.e., $r_k^{(i+1)} = 1$ if $M_{\psi(p)}$ is an inference task for DT k). Then, the related parameters are updated as follows:

$$\begin{aligned} z^{(i+1)} &= \lfloor (s_{\psi(p)} - t_{\mathcal{S}^{(i)}}^{\text{end}})/T_{\text{upd}}^{\text{comp}} \rfloor, \\ t_{\mathcal{S}^{(i+1)}}^{\text{start}} &= z^{(i+1)} T_{\text{upd}}^{\text{comp}}, \\ t_{\mathcal{S}^{(i+1)}}^{\text{end}} &= t_{\mathcal{S}^{(i+1)}}^{\text{start}} + T_{\text{inf}}^{\text{comp}}, \\ i &= i+1. \end{aligned}$$

On the other hand, if $\lfloor (t_{M_{\psi(p)}}^{\text{comp}} - t_{\mathcal{S}^{(i)}}^{\text{end}})/T_{\text{upd}}^{\text{comp}} \rfloor = 0$ that means any update tasks cannot be scheduled between $\mathcal{S}^{(i)}$ and $s_{\psi(p)}$, the scheduler inserts $M_{\psi(p)}$ into $\mathcal{S}^{(i)}$ and updates the termination time of $\mathcal{S}^{(i)}$ such that $t_{\mathcal{S}^{(i)}}^{\text{end}} = t_{\mathcal{S}^{(i)}}^{\text{end}} + T_{\text{inf}}^{\text{comp}}$.

According to the ASs, $z^{(i)}$ is equal to the number of update tasks that can be scheduled between $\mathcal{S}^{(i-1)}$ and $\mathcal{S}^{(i)}$. Therefore, we can formulate an integer programming problem $\mathcal{P}^{(i)}$ for $\mathcal{S}^{(i)}$ as follows:

$$\mathcal{P}^{(i)}: \max_{\boldsymbol{x}^{(i)}} \sum_{k=1}^{K} \sigma_k^{(i)} x_k^{(i)}, \tag{6}$$

$$\text{s.t. } \sum_{k=1}^{K} x_k^{(i)} \le z^{(i)}, \tag{6a}$$

$$\beta_k^{(i-1)} x_k^{(i)} = x_k^{(i)}, \forall k \in \mathcal{K}, \tag{6b}$$

where $\sigma_k^{(i)}$ is the value of scheduling $M_{k,0}$ in iteration i, $\boldsymbol{x}^{(i)} = [x_1^{(i)}, ..., x_K^{(i)}]^{\mathrm{T}}$ is the solution vector and $\beta_k^{(i-1)} \in \{0,1\}$ indicates whether $M_{k,0}$ has been scheduled in previous iterations. (6a) is the number of update tasks that can be scheduled and (6b) denotes the no-duplication constraint for each update task. Specifically, $\boldsymbol{\beta}^{(0)} = [\beta_1^{(0)}, ..., \beta_K^{(0)}]^{\mathrm{T}}$ is initialized as the vector of ones. According to $x_k^{(i)}$, $\beta_k^{(i)}$ is updated as follows:

$$\beta_k^{(i)} = \begin{cases} \beta_k^{(i-1)}, & \text{if } x_k^{(i)} = 0 \\ 0, & \text{if } x_k^{(i)} = 1. \end{cases} \tag{7}$$

Using ASs, a solution matrix $\boldsymbol{X} = [\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}]$ can be interpreted as a scheduler output $\boldsymbol{\varphi}$ where N is the total number

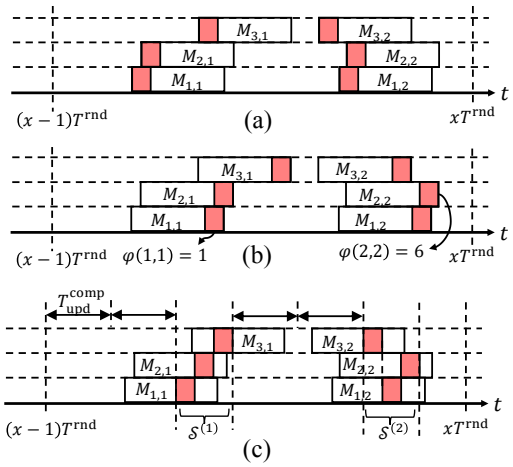

Fig. 4. An example of building ASs; (a) initial arrivals of inference tasks; (b) computation alignment to the deadlines; (c) building ASs using the computation delay of update tasks.

of ASs. Therefore, we can calculate the sum of freshness using $\boldsymbol{X}$. Let $\Theta(\boldsymbol{X})$ be the sum of freshness from a solution matrix $\boldsymbol{X}$ such that $\Theta(\boldsymbol{X}) = \sum_{k=1}^{K} F(k; \boldsymbol{X})$ where $F(k; \boldsymbol{X})$ is the freshness of DT k. Using the matrices $\boldsymbol{R} = [\boldsymbol{r}^{(1)}, ..., \boldsymbol{r}^{(N)}]$ and $\boldsymbol{B} = [\boldsymbol{\beta}^{(1)}, ..., \boldsymbol{\beta}^{(N)}]$, we can calculate $F(k; \boldsymbol{X})$ as follow:

$$F(k; \boldsymbol{X}) = \boldsymbol{r}_k^{(1)} \times (\bar{\boldsymbol{\beta}}_k^{(1)})^{\mathrm{T}}, \tag{8}$$

where $\boldsymbol{r}_k^{(i)}$ and $\boldsymbol{\beta}_k^{(i)}$ are the partial row vectors of $\boldsymbol{R}$ and $\boldsymbol{B}$, i.e., $\boldsymbol{r}_k^{(i)} = [r_k^{(i)}, ..., r_k^{(N)}]$ and $\boldsymbol{\beta}_k^{(i)} = [\beta_k^{(i)}, ..., \beta_k^{(N)}]$, respectively. $\bar{\boldsymbol{\beta}}_k^{(i)}$ is the ones' complement of $\boldsymbol{\beta}_k^{(i)}$.

To make the iterative integer programming problem in (6) equivalent to the maximization problem in (5), the value vector $\boldsymbol{\sigma}^{(i)}$ has to be sophisticatedly designed. From (8), the attainable freshness by scheduling $M_{k,0}$ at $\mathcal{S}^{(i)}$ is $\sum_{n=i}^{N} r_k^{(n)}$ since $\bar{\beta}_k^{(n)} = 1$, $\forall n \geq i$ if $x_k^{(i)} = 1$. Therefore, we can derive the following conditions for maximizing $\Theta(\boldsymbol{X})$:

- *Condition 1*: If $\boldsymbol{r}_{k1}^{(i)} = [1, ..., 1]$, $\sigma_{k1}^{(i)}$ must be the highest value.
- *Condition 2*: If $r_{k1}^{(i)} = 1$ and $r_{k2}^{(i)} = 0$, $\sigma_{k1}^{(i)}$ must be higher than $\sigma_{k2}^{(i)}$.

Condition 1 can be easily derived since satisfying Condition 1 results in the maximum attainable value of $\sum_{n=i}^{N} r_{k1}^{(n)} = N - i$. The validity of Condition 2 can be shown by using an example where $z^{(i)} = 1$, $\boldsymbol{r}_{k1}^{(i)} = [1, 0, ..., 0]$, and $\boldsymbol{r}_{k2}^{(i)} = [0, 1, ..., 1]$. If $x_{k2}^{(i)} = 1$, the achieved freshness at $\mathcal{S}^{(i)}$ will be $N - i - 1$ and there is no more attainable freshness from scheduling $M_{k1,0}$ in the future iterations. On the other hand, if $x_{k1}^{(i)} = 1$, the achieved freshness at $\mathcal{S}^{(i)}$ will be 1 but the scheduler can attain additional freshness whose amount is $N - i - 1$ at $\mathcal{S}^{(i+1)}$ because $\sigma_{k2}^{(i+1)}$ will be highest value by satisfying Condition 1. Finally, the amount of total freshness by assigning $x_{k1}^{(i)} = 1$ will be $N - i$. Therefore, $\sigma_k^{(i)}$ must include not only the value of the currently attainable freshness from $r_k^{(i)}$ (Condition 2) but also the freshness from $\boldsymbol{r}_k^{(i+1)}$ that is attainable in the future iterations (Condition 1). Let $v^{(i)} = (\boldsymbol{\beta}^{(i)})^{\mathrm{T}} \times \boldsymbol{r}^{(i)}$ which is the number of update tasks a) that can be scheduled at $\mathcal{S}^{(i)}$ and b) whose corresponding inference task is in $\mathcal{S}^{(i)}$. When $v^{(i)} = z^{(i)}$, Condition 2 is enough to decide $\boldsymbol{x}^{(i)}$ (i.e., $x_k^{(i)} = \beta_k^{(i-1)} r_k^{(i)}$). However, if $v^{(i)} < z^{(i)}$ or $v^{(i)} > z^{(i)}$, the scheduler must consider $\boldsymbol{r}_k^{(i+1)}$, $\forall k \in \mathcal{K}$. When assigning values to $r_k^{(i+1)}, ..., r_k^{(N)}$, Conditions 2 is recursively applied. Therefore, each of $r_k^{(i+1)}, ..., r_k^{(N)}$ must be assigned an exponentially decreasing value in the order of ASs for satisfying $\sigma_k^{(i+1)} < \sigma_k^{(i)} - \sigma_k^{(i+1)}$ when $r_k^{(i)} = 1$. Finally, we design $\sigma_k^{(i)}$ satisfying Condition 1 and Condition 2 as follows:

$$\sigma_k^{(i)} = \sum_{j=i}^{N} a^j r_k^{(j)}, \tag{9}$$

where a is used to give the exponentially decreasing value to $r_k^{(j)}$ in order of ASs and the proper range of a is given as $0 < a \leq 0.5$. Using (9), each integer programming problem in (6) can be easily solved by sorting $\boldsymbol{\sigma}^{(i)} = [\sigma_1^{(i)}, ..., \sigma_K^{(i)}]^{\mathrm{T}}$. The solution matrix $\boldsymbol{X}^*$ by solving $\mathcal{P}^{(i)}$ is optimal since it attains achievable freshness by satisfying Conditions 1 and 2. Also, the problems in (5) and (6) are equivalent since $\boldsymbol{X}^*$ can be transformed into the optimal permutation of DT tasks $\boldsymbol{\varphi}^*$.

The transformation from (5) into (6) extremely reduces the computational complexity. We can use an exhaustive search method for solving (5). However, the complexity of the exhaustive search will grow as $O(Q!)$ for searching every possible permutation of all DT tasks, which is computationally infeasible. On the other hand, the proposed scheme consists of two stages: building ASs and solving (6). When building ASs, the proposed scheme requires Q^2 comparisons, swaps, and arithmetic operations for sorting and searching all inference tasks. The complexity of solving (6) includes NK arithmetic operations for calculating $\sigma_k^{(i)}$ and NK^2 comparisons, swaps, and arithmetic operations for sorting update tasks at each AS. Therefore, the total complexity of the proposed scheme grows as $O(Q^2 + NK^2)$.

TABLE I
DEFAULT SIMULATION SETTING

Parameters	Value
Number of PTs (K)	10
Unit data size (δ)	1 Mbit
Sampling frequency (α)	10/round
Computational complexity of DT tasks (ω)	500 cycles/bit
Computational capability of the ES (f^{edge})	30 GHz
Minimum and maximum intervals ($[\phi^{\min}, \phi^{\max}]$)	$[0.7, 1.5]$s
Duration of a round (T^{rnd})	20s

IV. Simulation Results and Analysis

Extensive simulations are conducted to analyze the performance of our proposed scheme to solve the maximum freshness problem. Unless stated otherwise, the *default* simulation parameters are the ones summarized in Table I. Throughout the conducted experiments, we evaluate the performance of the proposed scheme by comparing it with a real-time task scheduling algorithm, named *earliest deadline first (EDF)* [11], which is our baseline. In EDF, the priority of inference tasks is higher than that of update tasks due to their shorter deadlines. Moreover, in EDF, the relative priority among inference tasks is assigned based on a first-in-first-out manner, whereas that among update tasks is assigned randomly. Next, we discuss the experiments' results.

*1) **Impact of Number of PTs on Total DT Freshness**:* First, we analyze the impact of increasing the number of PTs K in the DTEN and analyze its impact on the *total DT freshness*, defined as the sum of the achieved freshness for every DT in the DTEN (i.e., $\sum_{k=1}^{K} F(k, \boldsymbol{\varphi})$). The achieved total DT freshness by both algorithms is upper bounded by the total number of inference tasks. From Fig. 5, we observe that as K increases, the total DT freshness resulting from EDF initially increases but it starts diverging when the number of PTs becomes high (i.e., $25 \leq K \leq 30$). In such cases, e.g., $K = 30$, our proposed scheme achieves over two-fold improvement in the total DT freshness, compared to EDF. This is because the computations of update tasks tend to be delayed due to their longer deadline than the inference tasks in EDF. On the contrary, in the proposed scheme, the computations of

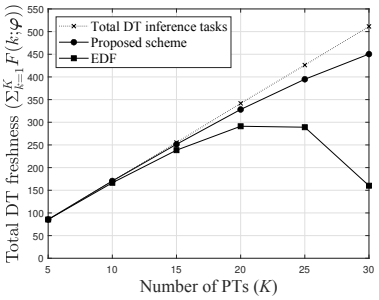

Fig. 5. Total DT freshness as a function of the number of PTs.

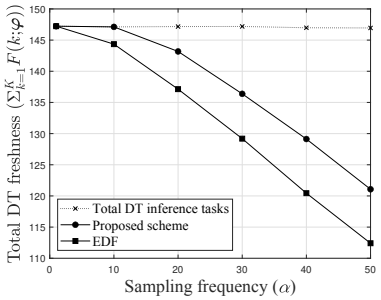

Fig. 6. Total DT freshness as a function of the number of samples for update task.

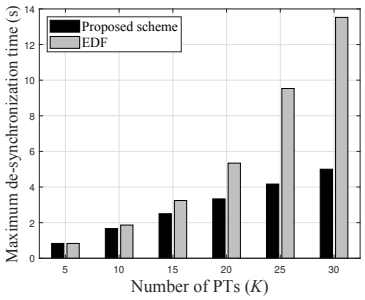

Fig. 7. Worst-case de-synchronization time according to the number of PTs.

inference tasks are delayed to schedule update tasks resulting in higher achieved total DT freshness than EDF. Moreover, the performance of our proposed scheme is near-optimal, where it is within 4% of the upper bound in the case of $K = 20$. However, in the extreme case of $K = 30$, this percentage slightly reduces to around 12%.

*2) **Impact of Number of Samples for Update Task on Total DT Freshness**:* Next, in Fig. 6, we analyze the total DT freshness when the sampling frequency α is varied. From Fig. 6, we observe that the total number of inference tasks remains steady as α varies since $\phi^{\min}$ and T^{rnd} are fixed. We also observe from Fig. 6 that for low sampling frequencies ($\alpha \leq 10$), by using our proposed scheme, an ES can accommodate more PTs and maintain its optimal freshness level. However, we observe that the total DT freshness achieved by the proposed scheme decreases as α increases (for $\alpha > 10$). This is due to the increased computation delay of the update tasks as α is increased. Similarly, the total DT freshness achieved by EDF decreases as α increases, even for very small sampling frequencies ($\alpha \leq 10$). This is due to the fact that increasing α results in a shorter duration where there are no inference tasks to be processed. However, the proposed scheme achieves a total DT freshness that is within around 8% of the total DT freshness achieved by EDF.

*3) **Impact of Number of PTs on Maximum De-synchronization Time**:* Finally, in Fig. 7, we study the impact of K on the *maximum de-synchronization time*, which is defined by the time when the computation of the last update task is completed. Obviously, increasing K always results in increasing the maximum de-synchronization time. We observe from Fig. 7 that the maximum de-synchronization time obtained from EDF increases exponentially while that resulting from the proposed scheme grows linearly as K increases. Fig. 7 demonstrates that the proposed scheme achieves a reduction of 63% in the maximum de-synchronization time compared to EDF when $K = 30$.

V. Conclusion

In this paper, we have investigated the real-time task scheduling problem in DTENs while distinguishing between the DT update and inference tasks. We have introduced a novel performance metric, freshness, to capture the number of inference tasks processed using the latest model. Moreover, to jointly fulfill the conflicting real-time requirements of those tasks, we have formulated a novel freshness maximization problem. We have proposed an efficient scheduling scheme to solve the formulated problem. To validate the effectiveness of the proposed scheme, we have conducted extensive experiments. Simulation results show that the total DT freshness achieved by the proposed scheme is within 4% of the upper bound with 20 PTs, and within 12% of the upper bound in worst cases of more than 30 PTs. Furthermore, simulation results have demonstrated that the proposed scheme can achieve a reduction of up to 63% in the maximum de-synchronization time compared to the EDF algorithm.

References

[1] O. Hashash, C. Chaccour, W. Saad, T. Yu, K. Sakaguchi, and M. Debbah, "The seven worlds and experiences of the wireless metaverse: Challenges and opportunities," *arXiv*, Apr. 2023. [Online]. Available: https://arxiv.org/abs/2304.10282

[2] L. U. Khan, W. Saad, D. Niyato, Z. Han, and C. S. Hong, "Digital-twin-enabled 6G: Vision, architectural trends, and future directions," *IEEE Commun. Mag.*, vol. 60, no. 1, pp. 74–80, 2022.

[3] O. Hashash, C. Chaccour, and W. Saad, "Edge continual learning for dynamic digital twins over wireless networks," in *Proc. IEEE Int. Workshop Signal Process. Advances Wireless Commun. (SPAWC)*, 2022, pp. 1–5.

[4] X. Lin, J. Wu, J. Li, W. Yang, and M. Guizani, "Stochastic digital-twin service demand with edge response: An incentive-based congestion control approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2402–2416, 2023.

[5] Y. Lu, S. Maharjan, and Y. Zhang, "Adaptive edge association for wireless digital twin networks in 6G," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16 219–16 230, 2021.

[6] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Low-latency federated learning and blockchain for edge association in digital twin empowered 6G networks," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 5098–5107, 2021.

[7] J. Wan, X. Li, H.-N. Dai, A. Kusiak, M. Martínez-García, and D. Li, "Artificial-intelligence-driven customized manufacturing factory: Key technologies, applications, and challenges," *Proc. IEEE*, vol. 109, no. 4, pp. 377–398, 2021.

[8] S. D. Okegbile, J. Cai, C. Yi, and D. Niyato, "Human digital twin for personalized healthcare: Vision, architecture and future directions," *IEEE Network*, pp. 1–7, 2022.

[9] 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.

[10] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, "Age of information: An introduction and survey," *IEEE J. Sel. Areas in Commun.*, vol. 39, no. 5, pp. 1183–1210, May 2021.

[11] M. Kargahi and A. Movaghar, "Non-preemptive earliest-deadline-first scheduling policy: A performance study," in *Proc. IEEE Int. Symp. Modeling, Anal., Simulation Comput. Telecommu. Syst.,*, 2005, pp. 201–208.