Workload-Efficient Deadline and Period Assignment for Maintaining Temporal Consistency under EDF

Jianjun Li, Ming Xiong, *Member*, *IEEE*, Victor C.S. Lee, *Member*, *IEEE*, LihChyun Shu, *Member*, *IEEE*, and Guohui Li

Abstract—Deriving deadlines and periods for update transactions so as to maintain timeliness and data freshness while minimizing imposed workload has long been recognized an important problem in real-time database research. Despite years of active research, the state-of-the-art still has much room for improvement, particularly for periodic transactions scheduled by the *Earliest Deadline First* (EDF) algorithm. In this paper, we propose a practical and efficient two-phase algorithm, *GEneral EDF* ($\mathcal{GE}_{EDF}$), for assigning periods and deadlines to EDF-scheduled update transactions. Phase 1 of $\mathcal{GE}_{EDF}$ aims at finding solutions for most inputs in linear time, based on the observation that the execution times of update transactions are relatively small compared to the validity interval lengths of real-time data objects in many real-time applications. In the remaining cases for which Phase 1 fails to derive solutions, Phase 2 is invoked by employing an existing deadline-monotonic-based algorithm, which we show is also applicable to our problem. Meanwhile, we have devised several techniques which significantly reduce the cost of schedulability test, and hence greatly improve time efficiency. Our experimental results demonstrate that $\mathcal{GE}_{EDF}$ outperforms existing approaches in terms of generated workloads. Although Phase 2 has a pseudopolynomial time complexity, our experimental study shows that it runs much faster than other solutions with comparable quality.

Index Terms—Real-time databases, temporal consistency, update transaction, periods and deadlines, earliest deadline first

1 Introduction

Real-time database systems (RTDBS) have been widely used in many applications that require timely processing of large amounts of real-time data, such as aerospace and defense systems, industrial automation and air traffic control systems. Typically, a real-time database is composed of a set of real-time data objects, each of which models the current status of a real-world entity in the external environment. Different from data stored in traditional databases, a real-time data object is only valid in a given time period, which is defined as its temporal validity interval. To maintain temporal validity (a.k.a. temporal consistency), each real-time data object must be refreshed by a sensor update transaction before its temporal validity interval expires, or else the RTDBS cannot respond to environmental changes in a timely manner.

Given the temporal consistency requirement, one important issue in designing RTDBS is to determine the sampling periods and deadlines for sensor update transactions so that temporal consistency can be maintained while the resulting processor workload is minimized. We call this the *period and deadline assignment problem*, which is important due to the following reasons [11], [30]: 1) it helps save sensor energy, because an inappropriate and unnecessarily short period of sensor update transactions may drive the sensor batteries flat quickly; 2) given the same portion of processor capacity, the RTDBS is able to process more sensor update transactions; and 3) system efficiency can be improved because more processor capacity can be left to other user transactions that are triggered due to environmental changes brought by sensor update transactions.

In resource-limited systems, like embedded systems, it is important for the software to maintain data freshness while minimizing the imposed workload. For instance, in the control units used for controlling the engine of a vehicle, assigning appropriate deadlines and periods to sensor transactions is crucial for timely monitoring of the condition of the engine operating environment, such as air pressure and engine temperature, so that the engine can be running in a fuel-efficient manner [7]. Another application example is the *age scheduling* that can be used to sample the signal flows through asynchronous distributed systems in a periodic manner [28]. We refer interested readers to [20] for more real-world examples of temporal consistency maintenance.

In the past, while there has been much work devoted to the period and deadline assignment problem, most of them

- *J. Li and G. Li are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Luoyu Road #1037, Wuhan 430074, Hubei, P.R. China. E-mail: jianjunli@smail.hust.edu.cn, guohuili@mail.hust.edu.cn.*
- *M. Xiong is with Google, Inc., 76 9th Avenue, New York, NY 10011. E-mail: mxiong@google.com.*
- *V.C.S. Lee is with the Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong. E-mail: csvlee@cityu.edu.hk.*
- *L. Shu is with the Department of Accountancy, College of Management, National Cheng Kung University, No. 1 University Road, Tainan 701, Taiwan, ROC, and the College of Information and Engineering, Chang Jung Christian University, Tainan, Taiwan, ROC. E-mail: shulc@mail.ncku.edu.tw.*

Manuscript received 18 Apr. 2011; revised 30 Dec. 2011; accepted 21 Feb. 2012; published online 7 Mar. 2012.
Recommended for acceptance by S. Dolev.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-04-0254.
Digital Object Identifier no. 10.1109/TC.2012.69.

are focused on *fixed priority* scheduling. Some examples are Half-Half (*HH*) [11], More-Less ($\mathcal{ML}_{DM}$) [30] and the deferrable scheduling algorithm for fixed-priority transactions (*DS-FP*) [9], [29]. One exception is the work conducted by Xiong et al. [31] who designed a linear-time EDF algorithm, called $\mathcal{ML}_{EDF}$, which solves the assignment problem for update transactions with deadlines no greater than their corresponding periods. $\mathcal{ML}_{EDF}$ is efficient in time, but is built on a sufficient (not necessary) feasibility condition, which means it may not produce optimal solutions that minimize imposed CPU workload for some inputs. The same authors further proposed $\mathcal{OS}_{EDF}$ [31], a *branch-and-bound*-based search algorithm, with the goal of obtaining optimal solutions. By assuming a discrete-time system, $\mathcal{OS}_{EDF}$ can handle transactions with arbitrary deadlines in relation to their periods. The main problem with $\mathcal{OS}_{EDF}$ is that it does not scale well with increasing problem size. To address this problem, they further proposed $\mathcal{HS}_{EDF}$ [31], a heuristic search-based algorithm, which is always capable of finding a solution if one exists. Compared to $\mathcal{OS}_{EDF}$, $\mathcal{HS}_{EDF}$'s efficiency is achieved at the expense of increased processor workload.

In this study, we take a fresh look at the problem of determining deadlines and periods for *firm periodic* real-time update transactions scheduled under EDF. Our aim is to continue to push the envelope and further advance the state of the art. We propose a general two-phase algorithm, called *GEneral EDF* (or $\mathcal{GE}_{EDF}$ for short), which outputs periods and deadlines for update transactions that result in significantly lower workload than that by existing approaches. The first phase of $\mathcal{GE}_{EDF}$ has a linear time complexity and hence can be utilized to derive a solution very efficiently. The second phase is invoked only when phase one fails. Though having a pseudopolynomial time complexity, the second phase can also run in a time-efficient manner due to our novel techniques which significantly reduce the overhead when checking schedulability. In a nutshell, the proposed algorithm is practical and efficient for many real-time applications. Specifically, $\mathcal{GE}_{EDF}$ is important in cases where a polynomial-time test produces transaction sets with relatively high workload, or an exact test runs unacceptably slow, though optimal in workload reduction. Our study shows that such cases are not uncommon. Comparing with previous work, our contributions can be summarized as follows:

1. We propose a novel approach based on *EDF*, namely $\mathcal{GE}_{EDF}$, which outperforms all existing periodic approaches based on *deadline monotonic (DM)* and *EDF* scheduling in terms of schedulability and CPU workload.
2. We introduce several techniques which significantly reduce the overhead when checking schedulability. As a result, $\mathcal{GE}_{EDF}$ can run in a time-efficient manner, although it has a pseudopolynomial time complexity.
3. We have conducted extensive simulation experiments to compare the performance of $\mathcal{GE}_{EDF}$ with existing periodic schemes. The experiment results demonstrate that $\mathcal{GE}_{EDF}$ is an efficient solution that can significantly reduce resulting CPU workload compared to existing approaches.

Organization: The remainder of this paper is organized as follows: Section 2 reviews the definition of temporal validity and presents some assumptions, as well as the period and deadline assignment problem to be addressed. Section 3 presents the details of the two phases of $\mathcal{GE}_{EDF}$. Section 4 presents the performance studies. Section 5 briefly discusses related work. Finally, Section 6 concludes the paper.

2 Preliminaries and Problem Statement

In this section, we first review the definition of temporal validity for data freshness, and then present some notations as well as important assumptions made throughout the paper. Finally, we precisely introduce the problem to be addressed.

2.1 Temporal Validity for Data Freshness

In a real-time database, a data object is a logic image of a real-world entity. As the state of a real-world entity changes continuously, to monitor the entity's state faithfully, real-time data objects must be refreshed by update transactions, which are generated periodically by intelligent sensors, before they become invalid. The actual length of the temporal validity interval of a real-time data object is application dependent [20], [22], [23]. We assume that a sensor always samples the value of a real-time data object at the beginning of its update period.

Definition 1. *A real-time data object (x_i) at time t is temporally valid if, for its jth update finished last before t, the sampling time ($r_{i,j}$) plus the validity interval ($\mathcal{V}_i$) of the data object is not less than t, i.e., $r_{i,j} + \mathcal{V}_i \geq t$.*

A value for real-time data object x_i sampled at any time t will be valid from t up to $(t + \mathcal{V}_i)$. To satisfy the validity constraint, for each x_i, the corresponding update transaction should execute at least twice during $\mathcal{V}_i$. Traditional methods for maintaining temporal validity, such as *HH* and *More Less*, have been proposed based on fixed-priority scheduling algorithms. Readers are referred to [11], [22], [30] for more details about these approaches.

2.2 Notations and Assumptions

In this paper, we use $\mathcal{T} = \{\tau_i\}_{i=1}^{n}$ and $\mathcal{X} = \{x_i\}_{i=1}^{n}$ to denote a set of periodic sensor update transactions and a set of real-time or temporal data, respectively. All temporal data are assumed to be kept in main memory. Each data $x_i (1 \leq i \leq n)$ is associated with a validity interval length $\mathcal{V}_i$. Transaction τ_i is responsible for updating the corresponding data x_i periodically. Since each sensor update transaction updates different data, no concurrency control is considered. Each update transaction τ_i is periodic and is characterized by the following 3-tuple: $\{C_i, D_i, T_i\}$, where C_i is the execution time, D_i is the relative deadline and T_i is the period. Here, we consider arbitrary deadlines where D_i can be less than, equal to, or larger than T_i. The utilization of τ_i is $U_i = \frac{C_i}{T_i}$, while the density factor (DF) of τ_i is $\lambda_i = \frac{C_i}{\mathcal{V}_i}$. Let $\mathcal{U}$ and λ denote the total processor utilization and the total DF of $\mathcal{T}$, respectively, i.e., $\mathcal{U} = \sum_{i=1}^{n} \frac{C_i}{T_i}$ and $\lambda = \sum_{i=1}^{n} \frac{C_i}{\mathcal{V}_i}$. We assume that the system is synchronous, i.e., the first instances of all sensor update transactions are generated at the same time

TABLE 1
Symbols and Definitions

Symbol	Definition
x_i	Real-time data object i
τ_i	Sensor update transaction updating x_i
$\mathcal{T}$	A set of update transactions, $\{\tau_1, \tau_2, \ldots, \tau_n\}$
$\mathcal{T}_k$	A subset of $\mathcal{T}$, $\{\tau_1, \tau_2, \ldots, \tau_k\}, 1 \leq k \leq n$
C_i	Execution time of τ_i
$\mathcal{V}_i$	Validity interval length of x_i
T_i	Period of τ_i
D_i	Relative deadline of τ_i
U_i	Processor workload of τ_i
$\mathcal{U}$	Processor workload of $\mathcal{T} = \{\tau_i\}_{i=1}^n$
λ	Density factor of $\mathcal{T} = \{\tau_i\}_{i=1}^n$

(e.g., time 0), and the jitter between sampling time and release time of a transaction instance is 0. Lastly, the scheduling algorithms studied in this work are considered preemptive. Table 1 presents the formal definitions of the symbols used in this work.

2.3 Problem Statement

Given a set of update transactions, the optimal solution to the deadline and period assignment problem must minimize the processor workload while maintaining temporal consistency under EDF. Consequently, the period and deadline assignment problem for real-time update transactions scheduled under EDF can be described as follows:

Deadline and period assignment problem (DPAP): *Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^n$ with C_i and $\mathcal{V}_i$ specified for each τ_i, determine T_i and D_i for τ_i so that the processor workload $\mathcal{U}(\mathcal{T})$ is minimized subject to the following constraints:*

- Validity constraint*: The sum of the period and relative deadline of transaction τ_i is not larger than the validity interval length $\mathcal{V}_i$, i.e., $T_i + D_i \leq \mathcal{V}_i$.*
- Schedulability constraint*: The transaction set $\mathcal{T}$ must be schedulable under EDF with derived deadlines $\{D_i\}_{i=1}^n$ and periods $\{T_i\}_{i=1}^n$.*

With EDF scheduling, Baruah et al. proposed an exact, albeit complex, schedulability test [2], which is later improved in [4], [32].

Theorem 1 [4], [32]. *Given a periodic task set $\mathcal{T} = \{\tau_i\}_{i=1}^n$, $\mathcal{T}$ is schedulable by EDF if and only if $\mathcal{U} \leq 1$ and $\forall t \in S$,*[1]

$$h(t,n)^1 = \sum_{j=1}^{n} \left(\left\lfloor \frac{t - D_j}{T_j} \right\rfloor + 1 \right) C_j \leq t, \tag{1}$$

where $S = \{d_k | d_k = kT_i + D_i \wedge d_k \leq \min(L_a^n, L_b^n)^1, k \in \mathbb{N}\}$,

$$L_a^n = \max\left\{ D_1, \ldots, D_n, \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - \mathcal{U}} \right\} \tag{2}$$

and L_b^n denotes the synchronous busy period (the length of the first processor busy period of the synchronous arrival pattern described in Definition 2 below).

Definition 2 [27]. *A synchronous busy period is a processor busy period in which all tasks are released simultaneously at the beginning of the processor busy period and ended by the first processor idle period (the length of such a period can be 0).*

1. Note that we use $h(t,n)$, rather than $h(t)$ as in [4], for the processor workload since the notation is used both here and later in Section 3.2.2 where we must consider different subsets of $\mathcal{T}$. For the same reason, we use L_a^n and L_b^n here, as opposed to L_a and L_b in [4].

By using the notations given in Theorem 1 and the theorem itself, the *Deadline and Period Assignment Problem* can be rephrased as follows:

DPAP. *Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^n$ with C_i and $\mathcal{V}_i$ specified for each τ_i, determine T_i and D_i for each τ_i in $\mathcal{T}$ such that $\mathcal{U}(\mathcal{T}) = \sum_{i=1}^n \frac{C_i}{T_i}$ is minimized subject to*

- Validity constraint*: $T_i + D_i \leq \mathcal{V}_i$.*
- Schedulability constraint*: $\forall t \in S, h(t,n) \leq t$.*

3 General Algorithm for the Assignment Problem

In this section, we first present our general algorithm $\mathcal{GE}_{EDF}$ by describing Phases 1 (denoted by $\mathcal{GE}_{EDF}^1$) and 2 (denoted by $\mathcal{GE}_{EDF}^2$) in Sections 3.1 and 3.2, respectively, and then discuss the relationship between Phases 1 and 2 in Section 3.3.

3.1 Phase 1: Finding a Solution in Linear Time

One might be tempted to solve *DPAP* directly. Solving this constrained optimization problem, however, can be extremely expensive, since verifying the constraint in (1) for all t values requires high computation complexity. In this phase, instead of tackling the assignment problem directly, we approach it circuitously, based on a theorem discussed shortly. We first present a useful lemma, which states a necessary condition for any task set to be schedulable under EDF [5].

Lemma 1. *Given a synchronous task set $\mathcal{T}$, let the tasks in $\mathcal{T}$ be sorted in nondecreasing order of deadlines (D_i) and suppose that the minimum task deadline, D_{min}, is unique. Regardless of the choices of periods, any task set that is schedulable under EDF must satisfy the following property:*

$$\sum_{i=1}^{j} C_i \leq D_j, \ \forall j = 1, \ldots, n. \tag{3}$$

To minimize the processor workload, T_i should be maximized, which in turn means D_i should be minimized, due to the validity constraint ($T_i + D_i \leq \mathcal{V}_i$). Based on Lemma 1, it is clear that we should set $D_i = \sum_{j=1}^{i} C_j$ and $T_i = \mathcal{V}_i - D_i$, which are assumed throughout Section 3.1. But notice that setting deadlines and periods this way does not necessarily guarantee schedulability. The following theorem characterizes a condition under which schedulability is guaranteed, which lays the foundation for $\mathcal{GE}_{EDF}^1$'s assignment of deadlines and periods.

Theorem 2. *Given an update transaction set $\mathcal{T} = \{\tau_i\}_{i=1}^n$ with deadlines derived by $D_i = \sum_{j=1}^{i} C_j$ and periods by $T_i = \mathcal{V}_i - D_i$, if the maximum deadline, i.e., $D_{\max} = \sum_{j=1}^{n} C_j$, is not larger than any period in $\mathcal{T}$, then $\mathcal{T}$ is guaranteed to be schedulable under EDF.*

Proof. Since $D_{\max} \leq T_k (1 \leq k \leq n)$, we know for each transaction $\tau_i (1 \leq i \leq n)$, there is $D_i \leq T_k (1 \leq k \leq n)$; hence, it is obvious that

$$C_i + \sum_{j=1}^{i-1} \lceil D_i/T_j \rceil C_j = \sum_{j=1}^{i} C_j = D_i,$$

which means $\mathcal{T}$ is schedulable under DM. Since any task (or transaction) set that is schedulable under DM is also schedulable under EDF [4], the theorem follows. □

It has been observed that in many real-time applications, the execution times of update transactions are relatively small compared with the validity interval lengths of real-time data objects [20], [30]. Based on this observation and our deadline and period assignment approach, the premise of Theorem 2 is likely to hold in many real applications. This is also demonstrated in our experimental study.

While Theorem 2 provides a useful sufficient condition for determining schedulability, it does not specify in what order transactions should be assigned deadlines and periods. It has been observed that different assignment orders can lead to different processor workloads. The *shortest validity first* (SVF) order, which assigns priorities to transactions in the inverse order of validity length with ties resolved in favor of transactions with less slack ($\mathcal{V}_i - C_i$ being the slack of τ_i), has been shown to be a good heuristic in many applications [3], [30], particularly when validity interval lengths are much larger than transaction computation times. This is also the story for our case. In fact, by using similar partition and merge operations as in [30], it can be shown that the solution with SVF is within $2\sum_{i\in\mathcal{T}} \left(\frac{C_i}{\mathcal{V}_i}\right)^2$ of that of an optimal solution, under similar restrictions. Considering that the validity interval lengths are much larger than corresponding transaction execution times, for example, avionics applications [11] with sensor transaction computation times in the range of milliseconds and validity interval lengths in the range of hundreds of milliseconds and seconds, this bound is actually very small and can be ignored. Since the derivation of the bound with SVF is quite similar to [30], we do not detail it in this work. Nevertheless, due to its near optimality, we will also use SVF in the following discussion unless otherwise specified, and leave the search for the general optimal assignment order as direction for future research.

Based on Theorem 2, we use the following approach to derive a solution to the assignment problem: For each transaction τ_i, we first set $D_i = \sum_{j=1}^{i} C_j$ and $T_i = \mathcal{V}_i - D_i$. Then, we check 1) $D_i \leq \frac{\mathcal{V}_i}{2}$,[2] 2) $D_{\max} \leq T_i$, and 3) $\mathcal{U} \leq 1$. If all these three conditions are satisfied, then D_i and T_i are determined, and we proceed to transaction τ_{i+1}. Otherwise, the process is terminated, meaning that $\mathcal{GE}^1_{EDF}$ fails to derive a solution. When each of the n transactions has been assigned a pair of deadline and period, $\mathcal{GE}^1_{EDF}$ succeeds in obtaining a solution. Algorithm 1 shows $\mathcal{GE}^1_{EDF}$ in pseudocode.

Algorithm 1. Phase 1 of $\mathcal{GE}_{EDF}$

1: **Input**: A set of update transactions $\mathcal{T} = \{\tau_i\}_{i=1}^n$ sorted in non-decreasing order of $\mathcal{V}_i$, with ties resolved in favor of transactions with less slack ($\mathcal{V}_i - C_i$);
2: **Output**: Deadlines $\{D_i\}_{i=1}^n$ and periods $\{T_i\}_{i=1}^n$;
3: $\mathcal{U} = 0$; $D_{\max} = \sum_{i=1}^n C_i$;
4: **for** $i = 1$; $i \leq n$; $i++$ **do**
5: $D_i = \sum_{j=1}^{i} C_j$; $T_i = \mathcal{V}_i - D_i$; $\mathcal{U} = \mathcal{U} + \frac{C_i}{T_i}$;
6: *// Check the conditions in Theorem 2;*
7: **if** $(D_i \leq \frac{\mathcal{V}_i}{2} \wedge D_{\max} \leq T_i \wedge \mathcal{U} \leq 1)$ **then**
8: continue;
9: **else**
10: abort;
11: **end if**
12: **end for**

Given a set of update transactions sorted in nondecreasing order of their validity interval lengths, $\mathcal{GE}^1_{EDF}$ has a time complexity of $O(n)$ and hence can be used to determine transaction deadlines and periods very efficiently. Moreover, whenever $\mathcal{GE}^1_{EDF}$ can derive a solution, the workload of this solution is always less than those achievable by all other schemes.

Theorem 3. *Given an update transaction set $\mathcal{T}$, if $\mathcal{GE}^1_{EDF}$ can derive a solution, then the workload of this solution is minimum over SVF ordering.*

Proof. Suppose there exists another feasible solution $\mathcal{A}$ under SVF ordering but with workload lower than that of $\mathcal{GE}^1_{EDF}$, then there must exist at least one transaction in $\mathcal{A}$ whose deadline is less than the one by $\mathcal{GE}^1_{EDF}$. Since the solution derived by $\mathcal{GE}^1_{EDF}$ is the one with minimum deadlines ($D_j = \sum_{i=1}^{j} C_j$) that can satisfy the property in Lemma 1, it is clear that $\mathcal{A}$ cannot satisfy the necessary condition in Lemma 1, and this contradicts the assumption that $\mathcal{A}$ is feasible. □

It should be noted that although $\mathcal{GE}^1_{EDF}$ can be utilized to derive solutions for a large number of cases efficiently, as demonstrated in our experimental study in Section 4, it still fails in certain cases, particularly when the number of transactions is large. This is because when determining D_i and T_i for τ_i, Phase 1 declares failure if there is $D_{max} > T_i$. When the scale of the transaction set is large, with the growth of the number of update transactions, the sum of the execution time, i.e., D_{max}, tends to exceed the derived period T_i, which means Phase 1 might fail in this case.

Nevertheless, we would like to stress that Phase 1 is not designed to handle all the cases. Rather, it services as a fast solution when the size of the given transaction set is not that large. Next, we introduce the second phase of $\mathcal{GE}_{EDF}$, which addresses the remaining cases for which $\mathcal{GE}^1_{EDF}$ fails to derive a solution.

3.2 Phase 2: Deriving a Solution Based on the Exact Schedulability Test in Theorem 1

As explained previously, we also use the SVF assignment order in this phase. We first utilize $\mathcal{ML}_{DM}$ to derive a preliminary solution, due to its time efficiency. Next, an enhanced solution with a lower processor workload is derived under EDF.

To minimize the update workload and guarantee temporal validity, $\mathcal{ML}_{DM}$ [30], a DM-based scheme, was proposed to schedule periodic update transactions. $\mathcal{ML}_{DM}$ derives a deadline for $\tau_i (1 \leq i \leq n)$ by finding the minimum solution of the recursive equation $\sum_{j=1}^{i} \lceil D_i/T_j \rceil C_j = D_i$, and then assigns a period to τ_i by $T_i = \mathcal{V}_i - D_i$, in SVF order. It

2. Notice here we require $D_i \leq \frac{\mathcal{V}_i}{2}$ since if $D_i > \frac{\mathcal{V}_i}{2}$, then $D_i > T_i$, which contradicts $D_i \leq D_{max} \leq T_i$.

TABLE 2
Parameters and Results for Example 1

i	C_i	$\mathcal{V}_i$	EDF		DM	
			D_i	T_i	D_i	T_i
1	3	16	3	13	3	13
2	4	16	7	9	7	9
3	5	46	19	27	23	23
Workload ($\mathcal{U}$)			0.86		0.893	

should be noted that although $\mathcal{ML}_{DM}$ has a pseudopolynomial time complexity, it is still time efficient in that it only takes $\mathcal{O}(\frac{\mathcal{V}_{max}}{2} \cdot n^2)$ time to derive a solution, where $\mathcal{V}_{max}$ is the maximum validity interval length and n is the number of transactions. While $\mathcal{ML}_{DM}$ only addresses deadline *constrained* transaction set where deadlines are no larger than their corresponding periods, another DM-based approach, $\mathcal{EML}_{DM}$, allows *arbitrary* deadlines which can be larger than, equal to, or less than the corresponding periods. Due to space limitations, readers are referred to [3], [31] for more details about $\mathcal{EML}_{DM}$.

Different from $\mathcal{ML}_{DM}$ and $\mathcal{EML}_{DM}$, in this work, we address how to guarantee temporal consistency under EDF. It is well known that EDF is optimal in the sense of feasibility on uniprocessor systems; we thus have the following result.

Theorem 4. *Given an update transaction set $\mathcal{T} = \{\tau_i\}_{i=1}^{n}$, if $\mathcal{ML}_{DM}$ can derive a solution to the period and deadline assignment problem, then this solution is also feasible under EDF.*

Proof. The claim follows directly from the optimality of EDF [4], i.e., any transaction set that is schedulable under DM is also schedulable under EDF. □

Theorem 4 provides us an alternative to deriving solutions to the assignment problem, since for a given transaction set, if we can obtain a solution by $\mathcal{ML}_{DM}$, then it means we have obtained a solution under EDF. In other words, we can employ $\mathcal{ML}_{DM}$ to obtain an initial solution which can then be further improved, i.e., the workload of this initial solution can be further decreased, provided the schedulability of the transaction set is still guaranteed. We illustrate this with an example below.

Example 1. Given an update transaction set with execution times and validity interval lengths as follows:

$$\tau_1 = (3, 16), \tau_2 = (4, 16), \tau_3 = (5, 46).$$

Solutions derived under EDF (by $\mathcal{OS}_{EDF}$ [31]; $\mathcal{OS}_{EDF}$ [31]; is a *branch-and-bound*-based search algorithm which can guarantee an optimal solution, as mentioned in Section 1) and DM (by $\mathcal{ML}_{DM}$) are shown in Table 2.

It can be seen that for the transaction set stated above, the solution derived under DM has a workload $\mathcal{U} = 0.893$, which is 3.3 percent larger than the workload achievable under EDF. The periods and deadlines of the transaction set derived under EDF and DM differ only in the third transaction. Based on the results, we can extend the period of the third transaction derived under DM, thus decreasing its deadline, and still guarantee schedulability under EDF, thereby reducing the processor workload. Now, the general question is for a given transaction set, if one can derive a solution under DM, is it always possible to find a solution under EDF with a workload no larger than the one achievable under DM? Fortunately, the answer is in the affirmative. We show below that when scheduling transactions sets with temporal consistency requirement, the best case of EDF-based solutions performs at least as good as the best case of DM-based solutions in terms of the generated processor workload. In our experimental evaluation in Section 4, our EDF-based solution is found to outperform DM-based ones in most of the cases.

Lemma 2. *Given an update transaction set $\mathcal{T}$, if $\mathcal{ML}_{DM}$ can derive a feasible solution, then the workload of this solution is the minimum achievable by any DM-based method.*

Proof. We prove it by contradiction. Suppose that there exists a feasible solution $\mathcal{A}$ under DM scheduling, but with workload lower than that of $\mathcal{ML}_{DM}$. Then, there must exist at least one transaction, say τ_k, in $\mathcal{A}$ with workload lower than the one derived by $\mathcal{ML}_{DM}$. Since the period and deadline are bounded by the validity interval length, we know that $D_k^{\mathcal{A}}$, the deadline of τ_k in $\mathcal{A}$ should be less than D_k, the one derived by $\mathcal{ML}_{DM}$, i.e., $D_k^{\mathcal{A}} < D_k$. Recall that $\mathcal{ML}_{DM}$ derives deadlines by finding the minimum solution of the recursive equation $D_i = \sum_{j=1}^{i} \lceil D_i/T_j \rceil C_j$. Moreover, the assignment order is fixed, it is obvious that $D_k^{\mathcal{A}}$ should be no less than D_k, i.e., $D_k^{\mathcal{A}} \geq D_k$. Otherwise, $\mathcal{A}$ is not feasible under DM. Hence, we come to a contradiction and the lemma follows. □

Theorem 5. *Given an update transaction set $\mathcal{T}$, the minimum workload derived under EDF is not larger than the minimum one under DM.*

Proof. Justified by the fact from Lemma 2 ($\mathcal{ML}_{DM}$ can result in the minimum workload solution under DM scheduling) and Theorem 4 (an $\mathcal{ML}_{DM}$ solution is also feasible under EDF). □

Based on Theorem 5, if one can derive a solution by $\mathcal{ML}_{DM}$, then it is possible to enhance the solution to further decrease the workload under EDF.

Another point we want to emphasize is that, based on Lemma 2 and Theorem 4, we know that every transaction set, for which a DM-based method is able to find a solution, also has a solution by an EDF-based method. However, the converse is not true. For example, suppose we reduce the validity interval length of τ_3 in Example 1 from 46 to be 38. $\mathcal{ML}_{DM}$ fails to derive a solution, even if we allow arbitrary transaction deadlines and use $\mathcal{EML}_{DM}$ to find a solution. But one can still obtain a solution under EDF (for example, by a search-based algorithm like $\mathcal{OS}_{EDF}$), with $D_3 = 19$ and $\mathcal{U} = 0.938$. This illustrates the superiority of EDF-based methods on solving the deadline and period assignment problem.

In summary, we know that when scheduling real-time update transactions, EDF outperforms DM not only on the range of transaction set, but also on the resulted processor workload. Nevertheless, in this work, we will utilize $\mathcal{ML}_{DM}$ to derive a preliminary solution (or part of the solution) at first, due to its time efficiency.

A discussion regarding the failure condition of $\mathcal{ML}_{DM}$: Since the maximum utilization that a transaction set can be schedulable is 100 percent and the DF λ should be less than the corresponding utilization ($\sum \frac{C_i}{\mathcal{V}_i} < \sum \frac{C_i}{T_i}$ because $T_i < \mathcal{V}_i$), one may wonder whether there exists any upper DF bound beyond which the transaction set is definitely unschedulable by $\mathcal{ML}_{DM}$? If we can find such an upper bound, then obviously, we have also identified the failure condition of $\mathcal{ML}_{DM}$. However, to our surprise, we found there does not exist such an upper bound, because even when λ approaches to 1, $\mathcal{ML}_{DM}$ can still derive a feasible solution sometimes. For example, given a transaction set with $n = 18$ and $\lambda_i (1 \le i \le 18)$ be 0.106, 0.073, 0.0718, 0.0709, 0.0669, 0.0628, 0.0608, 0.0537, 0.0505, 0.0452, 0.0419, 0.0374, 0.0308, 0.0301, 0.0301, 0.0129, 0.00854, and 0.0012, respectively, let K be a large integer (e.g., 1,000), we can construct the transaction set as follows:

- $T_1 = 1, C_1 = \frac{\lambda_1}{1-\lambda_1}, D_1 = C_1, \mathcal{V}_1 = D_1 + T_1, U_1 = C_1/T_1$;
- $T_2 = K$, $\mathcal{V}_2 = T_2/(1 - \lambda_2/(1 - U_1))$, $C_2 = \mathcal{V}_2\lambda_2$; $U_2 = C_2/T_2$, $D_2 = C_2/(1 - U_1)$;
  $\vdots$
- $T_i = K^i, \quad \mathcal{V}_i = T_i/(1 - \lambda_i/(1 - \sum_{j=1}^{i-1} U_j)), C_i = \mathcal{V}_i\lambda_i$; $U_i = C_i/T_i, D_i = C_i/(1 - \sum_{j=1}^{i-1} U_j)$.

It is not difficult to verify the schedulability of the constructed transaction set by performing response time analysis.[3] As the transaction set is a harmonic one, the schedulability upper bound is 100 percent. Following a similar analysis as in the above example, we can construct an $\mathcal{ML}_{DM}$ schedulable transaction set with a DF approaching 1 as n increases by using `Matlab`. Hence, the failure condition of $\mathcal{ML}_{DM}$ cannot be derived by merely judging the DF. In fact, we need to use $\mathcal{ML}_{DM}$ itself to check whether a transaction set is $\mathcal{ML}_{DM}$ schedulable or not, and this is exactly the method we have used in this work.

Depending on whether a solution can be obtained by $\mathcal{ML}_{DM}$ or not, Phase 2 can further be divided into two subcases, as detailed in the following.

3.2.1 Case 1: $\mathcal{ML}_{DM}$ Succeeds in Deriving a Solution

According to Theorem 4, we can obtain a feasible solution to the assignment problem. But according to Theorem 5, we know that the workload derived by $\mathcal{ML}_{DM}$ may be higher than the minimum one under EDF. Hence, as a second step, we need to enhance the solution to further decrease the workload without jeopardizing the schedulability.

To decrease workload and comply with the validity constraint, the only choice is to decrease one transaction's deadline, which inversely increasing its corresponding period. Since the assignment order is fixed, now our major concern is how to find the minimum deadline for τ_i, so that the EDF schedulability of the transaction set can be preserved.

Given a transaction τ_i, a simple but inefficient method to find the minimum deadline for τ_i would be starting by setting $D_i = C_i$, and then increasing D_i by one tick at each iteration, until the transaction set is found to be schedulable. This simple algorithm (denoted by one-tick increment scheme hereafter) is inefficient because it requires a large number of steps to terminate. In this work, we introduce a method to increment D_i by a suitable amount, which can significantly improve the efficiency. We will detail how to compute the increment amount later. At first, since the high complexity of utilizing an exact schedulability test comes from that one needs to check all the scheduling points, we introduce the following theorem to eliminate the unnecessary scheduling points when verifying schedulability at each iteration step. The initial value of D_i is set to be $D_{i-1} + C_i$, since according to Lemma 1, this is the minimum possible value that can guarantee schedulability.

Theorem 6. *For an EDF-schedulable update transaction set $\mathcal{T} = \{\tau_i\}_{i=1}^n$, when decreasing τ_i's deadline D_i to be D'_i and increasing T_i to be $T'_i = \mathcal{V}_i - D'_i$, one only needs to check the scheduling points in $\overline{S} = \{d_{i,k} | d_{i,k} = kT_j + D_j \wedge D'_i \le d_{i,k} \le D_i \cup d_{i,k} = D'_i, k \in \mathbb{N}, 1 \le j \le n \wedge j \ne i\}$ to verify the schedulability of the new transaction set.*

Proof. Given $\mathcal{T} = \{\tau_1, \ldots, \tau_i, \ldots, \tau_n\}$ which is schedulable under EDF, assume that we decrease D_i to be D'_i and at the same time increase T_i to be $T'_i = \mathcal{V}_i - D'_i$. Now, the new transaction set is $\mathcal{T}' = \{\tau_1, \ldots, \tau'_i, \ldots, \tau_n\}$ and the new set of scheduling points is

$$\begin{aligned} S' &= \{d_{i,k} | d_{i,k} = kT_j + D_j \wedge d_{i,k} \le \min(L'_a, L'_b) \cup d_{i,k} \\ &= kT'_i + D'_i \wedge d_{i,k} \le \min(L'_a, L'_b), k \in \mathbb{N}, 1 \\ &\le j \le n \wedge j \ne i\}, \end{aligned}$$

where L'_a and L'_b denote L_a and L_b computed based on the new transaction set, respectively. According to Theorem 1, we should check all the scheduling points in S' to verify the schedulability of $\mathcal{T}'$. However, as we will see, one only needs to check a small part scheduling points in S'. To facilitate distinction, we use $h'(t, n)$ to denote $h(t, n)$, but with D_i and T_i replaced by D'_i and T'_i, respectively.

Since $\mathcal{T}$ is schedulable under EDF, we know that for any time instance t (including all the scheduling points $d_{i,k}$), there is $h(t, n) \le t$. Given that $T_j + D_j = \mathcal{V}_j (1 \le j \le n)$, for all $d_{i,k} \in S'$, we have

$$\begin{aligned} h'(d_{i,k}, n) &= \sum_{j=1, j\ne i}^{n} \left\lfloor \frac{d_{i,k} + T_j - D_j}{T_j} \right\rfloor C_j + \left\lfloor \frac{d_{i,k} + T'_i - D'_i}{T'_i} \right\rfloor C_i \\ &= \sum_{j=1, j\ne i}^{n} \left(2 + \left\lfloor \frac{d_{i,k} - \mathcal{V}_j}{T_j} \right\rfloor\right) C_j + \left(2 + \left\lfloor \frac{d_{i,k} - \mathcal{V}_i}{T'_i} \right\rfloor\right) C_i. \end{aligned} \tag{4}$$

Since $T'_i \ge T_i$, if $d_{i,k} \ge \mathcal{V}_i$, we can derive that $\lfloor \frac{d_{i,k} - \mathcal{V}_i}{T'_i} \rfloor \le \lfloor \frac{d_{i,k} - \mathcal{V}_i}{T_i} \rfloor$, which further means $h'(d_{i,k}, n) \le h(d_{i,k}, n) \le d_{i,k}$. If $D_i < d_{i,k} < \mathcal{V}_i$, then $\lfloor \frac{d_{i,k} - \mathcal{V}_i}{T'_i} \rfloor = \lfloor \frac{d_{i,k} - \mathcal{V}_i}{T_i} \rfloor = -1$, and we have $h'(d_{i,k}, n) = h(d_{i,k}, n) \le d_{i,k}$. Hence, $h'(d_{i,k}) \le d_{i,k}$ always holds when $d_{i,k} > D_i$, which means we only need to check the scheduling points no larger than D_i. Moreover, notice that if $d_{i,k} < D'_i$, then

$$\left\lfloor \frac{d_{i,k} + T'_i - D'_i}{T'_i} \right\rfloor C_i = 0,$$

we hence know that $h'(d_{i,k}, n) = h(d_{i,k}, n) \le d_{i,k}$, which means we only need to verify $d_{i,k}$ no less than D'_i. For those scheduling points generated by decreasing D_i to be D'_i,

3. Note the above example may require some reduction of the value C_i (say Δ_i) such that $D_i = \sum_{j=1}^{i-1} \lceil D_i/T_j \rceil C_j + C_i - \Delta_i$ (to guarantee that the transaction set is DM schedulable). However, by choosing a sufficiently large K, Δ_i can be considered as a minor and negligible part.

i.e., $d_{i,k} = kT'_i + D'_i$, it is not difficult to see that only when $k = 0$, $d_{i,k} = D'_i$ satisfies $D'_i \le d_{i,k} \le D_i$. On the whole, we only need to check the points in $\overline{S} = \{d_{i,k} | d_{i,k} = kT_j + D_j \wedge D'_i \le d_{i,k} \le D_i \cup d_{i,k} = D'_i,\ k \in \mathbb{N},\ 1 \le j \le n \wedge j \ne i\}$ to verify the schedulability of $\mathcal{T}'$. □

By Theorem 6, we only need to check the scheduling points in $\overline{S}$ to verify schedulability, in contrast to the scheduling points in S defined in Theorem 1. It can be seen that $\overline{S}$, which is bounded by the range $[D'_i, D_i]$, is a small subset of S. Hence, the scheduling points can be significantly reduced. Now, we discuss how to compute the deadline increments by the following theorem.

Theorem 7. *Given an EDF-schedulable update transaction set $\mathcal{T} = \{\tau_i\}_{i=1}^n$, when decreasing τ_i's deadline D_i to be D'_i (and increasing T_i to be $T'_i = \mathcal{V}_i - D'_i$) and checking the schedulability of the new transaction set, once there exists a scheduling point $d_{i,k}$ with $h'(d_{i,k}, n)$[4] $> d_{i,k}$, then one can start the next iteration by setting $D'_i = h'(d_{i,k}, n)$.*

Proof. Suppose at $d_{i,k}$, $h'(d_{i,k}, n) > d_{i,k}$, i.e.,

$$\sum_{j=1, j\ne i}^{n} \left\lfloor \frac{d_{i,k} + T_j - D_j}{T_j} \right\rfloor C_j + \left\lfloor \frac{d_{i,k} + T'_i - D'_i}{T'_i} \right\rfloor C_i > d_{i,k}. \quad (5)$$

For transaction $\tau_j (1 \le j \le n, j \ne i)$, its period and deadline would not change at $d_{i,k}$, which also means the interference from τ_j would not change. Hence, to satisfy $h'(d_{i,k}, n) \le d_{i,k}$, D'_i has to be increased to reduce the interference caused by τ_i. To make sure at $t = d_{i,k}$, there is $h'(d_{i,k}, n) \le d_{i,k}$, we should start the next iteration by increasing D'_i to be larger than $d_{i,k}$ (notice here Theorem 6 guarantees that $D'_i \le d_{i,k} \le D_i$). Moreover, to ensure $h'(t) \le t$ for $t \in (d_{i,k}, h'(d_{i,k}, n)]$, D'_i also should be no less than $h'(d_{i,k}, n)$. Otherwise, suppose D'_i is increased to be D''_i which is less than $h'(d_{i,k}, n)$, and let $h''(t, n)$ denote $h(t, n)$ with D_i and T_i replaced by D''_i and T''_i, respectively, then there is

$$\begin{aligned} h''(D''_i, n) &= \sum_{j=1, j\ne i}^{n} \left\lfloor \frac{D''_i + T_j - D_j}{T_j} \right\rfloor C_j + \left\lfloor \frac{D''_i + T''_i - D''_i}{T''_i} \right\rfloor C_i \\ &= \sum_{j=1, j\ne i}^{n} \left\lfloor \frac{D''_i + T_j - D_j}{T_j} \right\rfloor C_j + \left\lfloor \frac{D''_i + T'_i - D'_i}{T'_i} \right\rfloor C_i \\ &= h'(D''_i, n). \end{aligned} \quad (6)$$

Since $d_{i,k} < D''_i < h'(d_{i,k}, n)$, it is not difficult to derive that $h'(D''_i, n) = h'(d_{i,k}, n) > D''_i$. By (6), it means $h''(D''_i, n) > D''_i$. Clearly, the transaction set remains unschedulable if D'_i is increased to be less than $h'(d_{i,k}, n)$. In summary, to guarantee all the scheduling points $t \in [d_{i,k}, h'(d_{i,k}, n)]$ satisfy $h'(t, n) \le t$, we should start the next iteration by increasing D'_i to be $h'(d_{i,k}, n)$. □

Theorems 6 and 7 allow us to derive the minimum deadline for τ_i efficiently due to the following two reasons. First, there is no need to check all the scheduling points at each iteration. Second, there is no need to test every possible deadline, which significantly reduces the number of iteration steps.

4. Note here $h'(d_{i,k}, n)$ denotes $h(d_{i,k}, n)$, which is defined in Theorem 1, but with D_i and T_i replaced by D'_i and T'_i, respectively.

3.2.2 Case 2: $\mathcal{ML}_{DM}$ Fails to Derive a Solution

In this case, since $\mathcal{ML}_{DM}$ cannot derive a solution directly, we first utilize $\mathcal{ML}_{DM}$ to derive a schedulable subset of $\mathcal{T}$, denoted as $\mathcal{T}_{k-1} (k \ge 2)$. The same as in Case 1, the processor workload of the solution to $\mathcal{T}_{k-1}$ may be higher than the minimum one which can be reached under EDF. Hence, we follow a similar method as in Case 1 to decrease the workload of $\mathcal{T}_{k-1}$. After that, we add τ_k into $\mathcal{T}_{k-1}$ and check whether $\mathcal{T}_k$ is schedulable to determine deadline and period for τ_k. Notice that in this process, the deadlines and periods in $\mathcal{T}_{k-1}$ remain unchanged. The initial deadline of τ_k is set to be $D_{k-1} + C_k$, since according to Lemma 1, this is the minimum possible value that can guarantee schedulability.

When checking the schedulability of $\mathcal{T}_k$, we introduce the following theorem to eliminate the unnecessary scheduling points at each iteration, and hence improve the efficiency.

Theorem 8. *When adding a new transaction τ_i into an EDF-schedulable transaction set $\mathcal{T}_{i-1}$, one only needs to check the scheduling points in $\overline{\overline{S}} = \{d_{i,k} | d_{i,k} = kT_j + D_j \wedge D_i \le d_{i,k} \le \min(L_a^i, L_b^i), k \in \mathbb{N}, 1 \le j < i\}$ to verify the schedulability of $\mathcal{T}_i$.*

Proof. Since $\mathcal{T}_{i-1}$ is schedulable under EDF, we have,

$$h(d_{i,k}, i-1) = \sum_{m=1}^{i-1} \left(\left\lfloor \frac{d_{i,k} - D_m}{T_m} \right\rfloor + 1 \right) C_m \le d_{i,k}. \quad (7)$$

Now, consider the two cases of τ_j:

1. $j = i$. It is apparent that $d_{i,k} = kT_i + D_i \ge D_i$.
2. $1 \le j < i$. If $d_{i,k} = kT_j + D_j < D_i$, then

$$\begin{aligned} h(d_{i,k}, i) &= \sum_{m=1}^{i} \left(\left\lfloor \frac{d_{i,k} - D_m}{T_m} \right\rfloor + 1 \right) C_m \\ &= \sum_{m=1}^{i-1} \left\lfloor \frac{d_{i,k} + T_m - D_m}{T_m} \right\rfloor C_m + \left\lfloor \frac{d_{i,k} + T_i - D_i}{T_i} \right\rfloor C_i \\ &= \sum_{m=1}^{i-1} \left\lfloor \frac{d_{i,k} + T_m - D_m}{T_m} \right\rfloor C_m = h(d_{i,k}, i-1) \le d_{i,k}, \end{aligned} \quad (8)$$

which means we only need to test those scheduling points no less than D_i.

In summary, one only needs to check the scheduling points in

$$\begin{aligned} \overline{\overline{S}} = \{&d_{i,k} | d_{i,k} = kT_j + D_j (1 \le j \le i) \wedge D_i \le d_{i,k} \\ &\le \min(L_a^i, L_b^i), k \in \mathbb{N}, 1 \le j < i\} \end{aligned}$$

to verify the schedulability of the new transaction set $\mathcal{T}_i$. □

Similar to Case 1, when determining the increment amount of D_i, we have the following theorem, which helps to decrease iteration steps and hence improves efficiency.

Theorem 9. *When adding a new update transaction τ_i into an EDF-schedulable transaction set $\mathcal{T}_{i-1}$ and checking the schedulability, if $h(d_{i,k}, i) > d_{i,k}$, one can start the next iteration by setting $D_i = h(d_{i,k}, i)$.*

Proof. Similar to reasoning in Theorem 7. □

If $\mathcal{T}_k$ is schedulable for deadlines and periods found for τ_k, we repeat the same process for τ_{k+1}. Finally, if all the remaining transactions $\tau_i (k < i \leq n)$ can pass the schedulability test with a pair of deadline and period, it means we have obtained a feasible solution to the assignment problem. Otherwise, $\mathcal{GE}^2_{EDF}$ declares failure, which means it is unable to derive a solution for the given transaction set.

Based on the two cases described above, we present the detail of $\mathcal{GE}^2_{EDF}$ in Algorithm 2. At first, we invoke $\mathcal{ML}_{DM}$ to see whether one solution can be found. If the answer is positive, we proceed to Case 1 to improve the solution derived by $\mathcal{ML}_{DM}$, as shown in lines 5-20. In this case, we first skip those transactions with $D_i = \sum_{j=1}^{i} C_j$ (lines 6,7), since according to Lemma 1, the deadline of these transactions cannot be decreased. Then, for each transaction τ_i, we set the initial value of D'_i to be $D_{i-1} + C_i$ and check the scheduling points in $\overline{S}$ according to Theorem 6. If for all the scheduling points in $\overline{S}$, $h'(d_{i,k}, n) \leq d_{i,k}$ holds, we thus obtain the minimum deadline for τ_i and proceed to next transaction τ_{i+1}. Otherwise, we increase the deadline according to Theorem 7 (lines 16, 17) and come to next iteration. If D'_i is finally found to be equal to D_i, it means that the deadline of τ_i remains unchanged, and obviously, the transaction set remains to be schedulable.

Algorithm 2. Phase 2 of $\mathcal{GE}_{EDF}$

1: **Input**: The same as in $\mathcal{GE}^1_{EDF}$;
2: **Output**: Deadlines $\{D_i\}_{i=1}^n$ and periods $\{T_i\}_{i=1}^n$;
3: Invoke $\mathcal{ML}_{DM}$;
4: **if** One solution can be derived **then**
5: **for** $(i = 1; i \leq n; i++)$ **do**
6: **if** $D_i = \sum_{j=1}^{i} C_j$ **then**
7: continue;
8: **end if**
9: **for** $(D'_i = D_{i-1} + C_i; D'_i \leq D_i;)$ **do**
10: $T'_i = \mathcal{V}_i - D'_i$;
11: $\overline{S} = \{d_{i,k} | d_{i,k} = kT_j + D_j \wedge D'_i \leq d_{i,k} \leq D_i \cup d_{i,k} = D'_i, k \in \mathbb{N}, 1 \leq j \leq n \wedge j \neq i\}$;
12: **if** $\forall d_{i,k} \in \overline{S}, h'(d_{i,k}, n) \leq d_{i,k}$ **then**
13: $D_i = D'_i$; $T_i = \mathcal{V}_i - D_i$;
14: break;
15: **else**
16: Find the first $h'(d_{i,k}, n) > d_{i,k}$;
17: $D'_i = h'(d_{i,k}, n)$;
18: **end if**
19: **end for**
20: **end for**
21: **else**
22: Find $\mathcal{T}_{k-1}$ which is schedulable by $\mathcal{ML}_{DM}$;
23: Repeat the steps in 5-20 (replace n with $k-1$) to adjust transaction deadlines and periods in $\mathcal{T}_{k-1}$;
24: **for** $(i = k; i \leq n; i++)$ **do**
25: **for** $(D_i = D_{i-1} + C_i; D_i \leq \mathcal{V}_i - C_i;)$ **do**
26: $\underline{T_i} = \mathcal{V}_i - D_i$;
27: $\overline{\overline{S}} = \{d_{i,k} | d_{i,k} = kT_j + D_j \wedge D_i \leq d_{i,k} \leq \min(L^i_a, L^i_{\underline{b}}), k \in \mathbb{N}, 1 \leq j < i\}$;
28: **if** $\forall d_{i,k} \in \overline{\overline{S}}, h(d_{i,k}, i) \leq d_{i,k}$ **then**
29: break;
30: **else**
31: Find the first $h(d_{i,k}, i) > d_{i,k}$;
32: $D_i = h(d_{i,k}, i)$;
33: **end if**
34: **end for**
35: **end for**
36: **end if**

TABLE 3
Parameters and Results for Example 2

i	C_i	$\mathcal{V}_i$	$\mathcal{ML}_{DM}$		$\mathcal{GE}^{II}_{EDF}/\mathcal{HS}_{EDF}$	
			D_i	T_i	D_i	T_i
1	3	15	3	12	3	12
2	4	16	7	9	7	9
3	5	48	23	25	19	29
Workload ($\mathcal{U}$)			0.894		0.867	

On the contrary, if no solution can be found by $\mathcal{ML}_{DM}$ directly, we proceed to the second case of Phase 2, as shown in lines 22-35. In this case, we first employ $\mathcal{ML}_{DM}$ to derive a schedulable subset $\mathcal{T}_{k-1}$. Then, following a similar method as in Case 1, we decrease the workload of $\mathcal{T}_{k-1}$. After that, we add one transaction (τ_k) into $\mathcal{T}_{k-1}$ at each step, and check whether the new transaction set, $\mathcal{T}_k$, is schedulable to determine a pair of deadline and period for τ_k. Note that here $\mathcal{V}_k - C_k$ is used to bound D_k due to the requirement that $C_k \leq \min(D_k, T_k)$. If for all the scheduling points in $\overline{\overline{S}}$, $h(d_{i,k}, i) \leq d_{i,k}$ holds, we thus obtain the minimum deadline of τ_k, and proceed to next transaction τ_{k+1}. Otherwise, Theorem 9 is invoked to determine the increment amount of D_k and start the next iteration.

In the following, we present two examples to illustrate the computation process of $\mathcal{GE}^2_{EDF}$. We first show an example where $\mathcal{ML}_{DM}$ succeeds in deriving a solution.

Example 2. Given a set of transactions with execution times and validity interval lengths as follows:

$$\tau_1 = (3, 15), \tau_2 = (4, 16), \tau_3 = (5, 48).$$

Since $\mathcal{ML}_{DM}$ can obtain a solution for this transaction set, as shown in Table 3, we enter Case 1 to further improve the solution (by decreasing the derived deadlines and hence reducing workload). Given that $D_1 = C_1$ and $D_2 = C_1 + C_2$, it is apparent that τ_1 and τ_2's deadline cannot be further decreased, and we thus proceed to τ_3. We set $D'_3 = D_2 + C_3 = 12$ to start the iteration (line 9). By Theorem 6, we have $\overline{S} = \{12, 15, 16\}$. It can then be examined that when $d_{i,k} = 16$, there is $h'(d_{i,k}, n) = h'(16, 3) = 19 > 16$. Since this is the first $h'(d_{i,k}, n) > d_{i,k}$, we set $D'_3 = h'(16, 3) = 19$ to start the next iteration. Finally, the transaction set is found to be schedulable with $D_3 = 19$ and $T_3 = 29$, and all the periods and deadlines are thus determined.

It can be seen that for the given transaction set, $G\mathcal{E}^2_{EDF}$ can result in a 2.7 percent workload reduction compared to $\mathcal{ML}_{DM}$. Notice that in this example, $\mathcal{HS}_{EDF}$ produces the same result as $G\mathcal{E}^2_{EDF}$. Next, we present another example where $\mathcal{ML}_{DM}$ fails to derive a solution.

Example 3. Given a set of transactions with execution times and validity interval lengths as follows:

TABLE 4
Parameters and Results for Example 3

i	C_i	$\mathcal{V}_i$	$\mathcal{HS}_{EDF}$		$\mathcal{GE}^{II}_{EDF}$	
			D_i	T_i	D_i	T_i
1	2	16	2	14	2	14
2	7	30	17	13	9	21
3	6	33	8	25	17	16
Workload ($\mathcal{U}$)			0.921		0.851	

$$\tau_1 = (2, 16), \tau_2 = (7, 30), \tau_3 = (6, 33).$$

Since $\mathcal{ML}_{DM}$ fails to derive a solution for this transaction set, we proceed to Case 2 and derive a schedulable subset $\{\tau_1, \tau_2\}$ with $D_1 = 2$, $T_1 = 14$, $D_2 = 9$, and $T_2 = 21$. Given $D_1 = C_1$ and $D_2 = C_1 + C_2$, it is obvious that D_1 and D_2 cannot be decreased any more. We then set the initial value of τ_3's deadline to be $D_3 = D_2 + C_3 = 15$ and check whether the transaction set is schedulabe with $D_3 = 15$ and $T_3 = 18$. It can be derived that when $t = T_1 + D_1 = 16$, there is $h(t, n) = h(16, 3) = 17 > 16$. Therefore, we start the next iteration by setting $D_3 = 17$. Finally, the transaction set is found to be schedulable with $D_3 = 17$ and $T_3 = 16$. The final solutions by $\mathcal{HS}_{EDF}$ and $\mathcal{GE}^2_{EDF}$ are stated in Table 4. Notice here $\mathcal{HS}_{EDF}$ does not follow the SVF assignment order.

It can be seen that for the given transaction set, $\mathcal{GE}^2_{EDF}$ obtains a solution with workload 0.851, which is significantly lower than 0.921, the one derived by $\mathcal{HS}_{EDF}$. The reason is as a heuristic search-based algorithm, $\mathcal{HS}_{EDF}$'s efficiency is achieved at the expense of increased processor workload.

$\mathcal{GE}^2_{EDF}$ has a pseudopolynomial time complexity. However, it runs much faster than the one which uses an exact schedulability test in the average case. The main reason is that $\mathcal{ML}_{DM}$ can cover a large scope of transaction sets, which implies the number of transactions that need to be added into the subset is small. Moreover, we have introduced four theorems (Theorems 6, 7, 8, and 9), which significantly reduce the scheduling points and decrease the number of iteration steps when verifying schedulability. All these factors make Phase 2 run in a time-efficient manner, as demonstrated in our experiments.

3.3 Relationship between Phases 1 and 2

It is important to note that Phase 2 covers a bigger range of transaction set than Phase 1, as stated below.

Theorem 10. *Given any set of update transactions $\mathcal{T}$, if Phase 1 can obtain a solution, then Phase 2 can also derive one, with the same CPU workload.*

Proof. If Phase 1 can derive a solution, it means $\mathcal{ML}_{DM}$ can also derive a solution for $\mathcal{T}$. According to the assignment scheme of $\mathcal{ML}_{DM}$, we know that the solution by $\mathcal{ML}_{DM}$ is the same as that of Phase 1. Obviously, the deadlines in this solution cannot be further decreased, which means the solution by Phase 2 is the same as that of $\mathcal{ML}_{DM}$, which in turn is the same as in Phase 1. □

Although Phase 2 can schedule a larger number of transaction sets than Phase 1, Phase 1 is still useful owing to its high time efficiency. In other words, the introduction of Phase 1 does not affect the time complexity of $\mathcal{GE}_{EDF}$.

TABLE 5
Experimental Parameters and Settings

Para. Class	Parameters	Meaning	Value
System	N_{CPU}	No. of CPU	1
	N_T	No. of data objects	[50,300]
	$\mathcal{V}_i$(ms)	Validity interval of x_i	[4000,8000]
Update Transactions	C_i(ms)	Time for updating x_i	[5,15]
	Trans.length	No. of data to update	1

4 Performance Evaluation

This section presents the performance evaluation of the proposed algorithms: $\mathcal{GE}_{EDF}$, versus existing approaches *HH* [11], $\mathcal{ML}_{DM}$ [30], $\mathcal{EML}_{DM}$ [3], [31], $\mathcal{ML}_{EDF}$ and $\mathcal{HS}_{EDF}$[5] [31] via simulation experiments. Section 4.1 describes the simulation model and parameters. Section 4.2 discusses the experimental results.

4.1 Simulation Model and Assumptions

We have conducted quantitative experiments to compare the performance of $\mathcal{GE}_{EDF}$ with *HH*, $\mathcal{ML}_{EDF}$, $\mathcal{ML}_{DM}$, $\mathcal{EML}_{DM}$, and $\mathcal{HS}_{EDF}$. Among these approaches, $\mathcal{ML}_{EDF}$ and $\mathcal{HS}_{EDF}$ are EDF-based schemes, whereas $\mathcal{ML}_{DM}$ and $\mathcal{EML}_{DM}$ are DM-based ones. *HH* can be either EDF based or DM based. While *HH*, $\mathcal{ML}_{DM}$, and $\mathcal{ML}_{EDF}$ only address transaction set with deadlines no larger than their corresponding periods, $\mathcal{EML}_{DM}$, $\mathcal{HS}_{EDF}$, and $\mathcal{GE}_{EDF}$ can handle arbitrary deadlines which are less than, equal to or larger than their corresponding periods. The update transaction workloads produced by these algorithms, as well as the execution times under these algorithms, have been compared. It is demonstrated that $\mathcal{GE}_{EDF}$ can result in a significantly reduced CPU workload which is lower than all existing approaches, in a time-efficient manner.

Table 5 shows a summary of the parameters and default settings used in our experiments. We use the same baseline values for the parameters as [31], which are originally from air traffic control applications [11], [20], for the following reasons: 1) To enable easy comparison and continuity with the several previous studies that have used similar models and parameter values; 2) Our objective is to evaluate the relative performance characteristics of the approaches, not their absolute levels.

Two categories of parameters are defined: system and update transaction. For system configurations, a single CPU, main memory-based RTDBS is considered. The number of real-time data objects N_T ranges from 50 to 300 to generate different workloads in the system. The validity interval length $\mathcal{V}_i$ of each real-time data object is assumed to be uniformly distributed in $[4,000, 8,000]$. For update transactions, it is assumed that each update transaction updates one data object, and the execution time of each transaction is uniformly distributed in $[5, 15]$. Under the default setting, Phase 1 can cover all the cases. To show the performance of Phase 2 of $\mathcal{GE}_{EDF}$, we vary the ranges of validity interval length and execution time to $[2,000, 14,000]$ and $[8, 18]$, respectively, to produce those cases that can only be scheduled by Phase 2. Moreover, to produce those cases

5. Since $\mathcal{OS}_{EDF}$ does not scale well with problem size, mentioned as in [31], we do not consider it in our experiments.

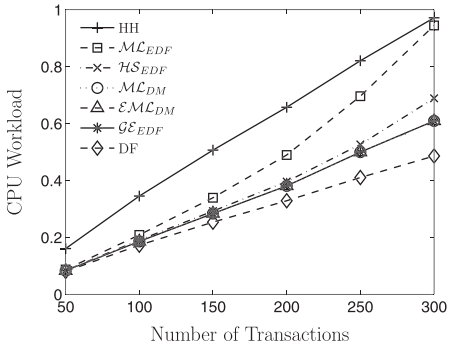

Fig. 1. Resulted workloads with $\mathcal{V}_i$ in $[4,000, 8,000]$ and C_i in $[5, 15]$.

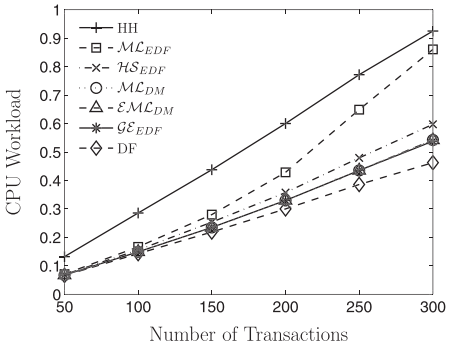

Fig. 2. Resulted workloads with $\mathcal{V}_i$ in $[2,000, 14,000]$ and C_i in $[5, 15]$.

that can show significant workload-difference between EDF-based and DM-based schemes, we also change the ranges of validity interval length and execution time to $[4,000, 16,000]$ and $[10, 20]$, respectively. It is noteworthy that here by varying execution times and validity interval lengths, we just want to investigate the scalability of our approach in terms of the workload caused by the update transactions. Besides the update transactions, the system also runs other tasks (user or triggered transactions); hence, it must properly control the update workload.

The proposed algorithms are all implemented in `C++`, while the knapsack problem in $\mathcal{HS}_{EDF}$ is solved by `Matlab`. For each point plotted in the figure, the simulations continued until a confidence interval of 95 percent with half-width of less than 5 percent about the mean was achieved.

4.2 Experimental Results

4.2.1 Comparison of CPU Workloads

The CPU workloads of update transactions produced by *HH*, $\mathcal{ML}_{EDF}$, $\mathcal{ML}_{DM}$, $\mathcal{EML}_{DM}$, $\mathcal{HS}_{EDF}$, and $\mathcal{GE}_{EDF}$ are quantitatively compared. The DF, which provides a lower bound of the workload, is also plotted. Three sets of experiments with different parameter settings have been conducted. In the first set, update transactions are generated randomly according to the parameter settings in Table 5. The resulting processor workloads from the six schemes are depicted in Fig. 1, in which the x-axis denotes the number of update transactions and the y-axis denotes the resulted CPU workloads.

It can be seen that $\mathcal{GE}_{EDF}$ consistently outperforms the three EDF-based schemes, i.e., *HH*, $\mathcal{ML}_{EDF}$, and $\mathcal{HS}_{EDF}$, as shown in Fig. 1. With the growth of N_T, the discrepancy between *HH*, $\mathcal{ML}_{EDF}$, and $\mathcal{HS}_{EDF}$ with $\mathcal{GE}_{EDF}$ increases. When N_T grows up to 300, the performance improvement of $\mathcal{GE}_{EDF}$ over the other three existing EDF-based approaches increases to 37, 34, and 8 percent, respectively. Given the small execution time and relatively large validity interval length in the default setting, $\mathcal{GE}_{EDF}$ can obtain a solution in Phase 1 while N_T varies from 50 to 300.

In the second set of experiments, we vary the range of validity interval length of x_i to $[2,000, 14,000]$. Other parameters remain the same as in Table 5. Fig. 2 presents the performance of $\mathcal{GE}_{EDF}$ as compared with the other five schemes. It can be observed that, similar to Fig. 1, $\mathcal{GE}_{EDF}$ consistently outperforms the other three EDF-based algorithms. When $N_T \geq 250$, $\mathcal{GE}_{EDF}$ fails to derive a solution in Phase 1, but can derive a solution in Phase 2. This is because when N_T exceeds 250, the sum of all transactions' execution times can be larger than the minimal period and therefore Phase 1 cannot derive a solution. But since $\mathcal{ML}_{DM}$ can still derive one, Phase 2 is also able to derive a solution when N_T varies from 250 to 300 in Fig. 2.

Fig. 3 presents the result of the third experiment set, in which the range of validity interval remains to be $[2,000, 14,000]$ but execution time varies uniformly between 8 and 18 ms, while other parameters remain the same as in Table 5. When $N_T \leq 150$, Phase 1 is able to derive a solution. When N_T varies between 150 and 300 in Fig. 3, Phase 2 can derive a solution after Phase 1 fails to do so. When $N_T \geq 230$, both *HH* and $\mathcal{ML}_{EDF}$ fail to derive a solution, because the DF λ becomes larger than 0.5. Similar to previous results, the workload derived by $\mathcal{GE}_{EDF}$ is consistently lower than all the other three EDF-based schemes. When N_T increases to 300, the improvement of $\mathcal{GE}_{EDF}$ over $\mathcal{HS}_{EDF}$ is about 10 percent.

For all the three experimental settings, the results of two DM-based *More-Less* schemes, i.e., $\mathcal{ML}_{DM}$ and $\mathcal{EML}_{DM}$ are also depicted in the three figures. As observed, the workloads of $\mathcal{ML}_{DM}$ and $\mathcal{EML}_{DM}$ are almost the same as that of $\mathcal{GE}_{EDF}$. In fact, in the first and second set of experiments, the workloads derived by $\mathcal{ML}_{DM}$ and $\mathcal{EML}_{DM}$ are exactly the same with that of $\mathcal{GE}_{EDF}$. This is mainly because given the range of validity interval lengths and transactions execution times in Table 5, most transaction deadlines in $\mathcal{ML}_{DM}$, $\mathcal{EML}_{DM}$, and $\mathcal{GE}_{EDF}$ are essentially derived by computing the sum of execution times of transactions with higher priorities, which means most (or all, when the number of transactions is small) transactions have the same workload.

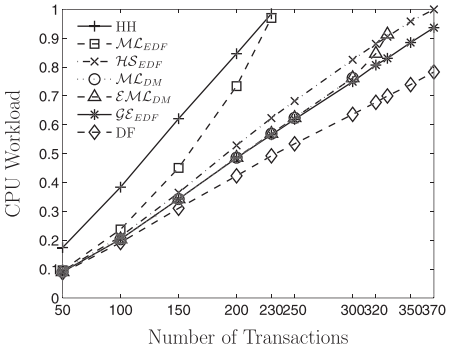

Fig. 3. Resulted workloads with $\mathcal{V}_i$ in $[2,000, 14,000]$ and C_i in $[8, 18]$.

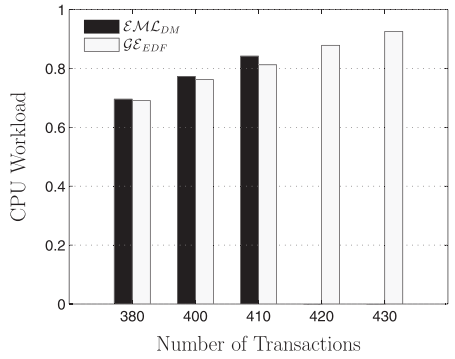

Fig. 4. $\mathcal{EML}_{DM}$ versus $\mathcal{GE}_{EDF}$ with $\mathcal{V}_i$ in $[2,000, 14,000]$ and C_i in $[5, 15]$.

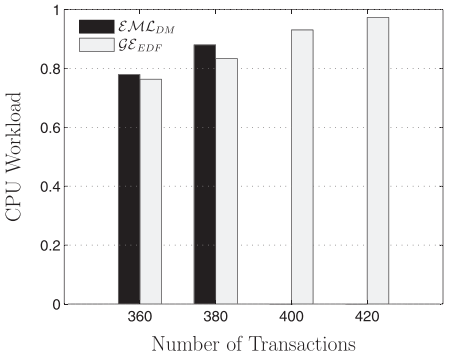

Fig. 6. $\mathcal{EML}_{DM}$ versus $\mathcal{GE}_{EDF}$ with $\mathcal{V}_i$ in $[4,000, 16,000]$ and C_i in $[10, 20]$.

In the third set of experiments, we observe that $\mathcal{ML}_{DM}$ fails to derive a solution when N_T exceeds 300, while $\mathcal{EML}_{DM}$ can still derive one until the number of transactions reaches 320, but with workload higher than that of $\mathcal{GE}_{EDF}$. Compared to $\mathcal{ML}_{DM}$, $\mathcal{EML}_{DM}$ can schedule a larger set of transactions. But when N_T exceeds 330, $\mathcal{EML}_{DM}$ also fails to derive a solution, whereas $\mathcal{GE}_{EDF}$ and $\mathcal{HS}_{EDF}$ can still derive one. As can be seen, both $\mathcal{GE}_{EDF}$ and $\mathcal{HS}_{EDF}$ can lead to a solution even when N_T reaches 370, which improve the schedulability of $\mathcal{EML}_{DM}$ by 12 percent. This illustrates that EDF-based *More-Less* approaches can schedule a broader set of update transactions than DM-based ones.

From Figs. 1, 2, and 3, we can see that the performance levels of $\mathcal{ML}_{DM}$, $\mathcal{EML}_{DM}$, and $\mathcal{GE}_{EDF}$ are close to each other for most cases, except in Fig. 3, where the performance gap between $\mathcal{GE}_{EDF}$ and $\mathcal{EML}_{DM}$ becomes significant (about 5 percent) when the number of transactions increases to 320. This is mainly because that $\mathcal{ML}_{DM}$, $\mathcal{EML}_{DM}$, and $\mathcal{GE}_{EDF}$ tend to produce optimal (or near optimal) solutions when the scale of the transaction set is relatively small. To further investigate their performance when the scale of transaction set is large, we also conduct another three set of experiments, in which the range of validity interval and execution time are located in $\{[2{,}000, 14{,}000], [5, 15]\}$, $\{[2{,}000, 14{,}000], [10, 20]\}$, and $\{[4{,}000, 16{,}000], [10, 20]\}$, respectively. Note here that by varying execution times and validity interval lengths, we just want to generate those cases where the differences between $\mathcal{EML}_{DM}$ and $\mathcal{GE}_{EDF}$ are more significant. Since $\mathcal{EML}_{DM}$ dominates $\mathcal{ML}_{DM}$, we only compare $\mathcal{EML}_{DM}$ with $\mathcal{GE}_{EDF}$. We also do not present the workload comparison when the scale of the transaction set is small since in this case the gap between $\mathcal{GE}_{EDF}$ and $\mathcal{EML}_{DM}$ is so tiny that they are hard to be distinguished.

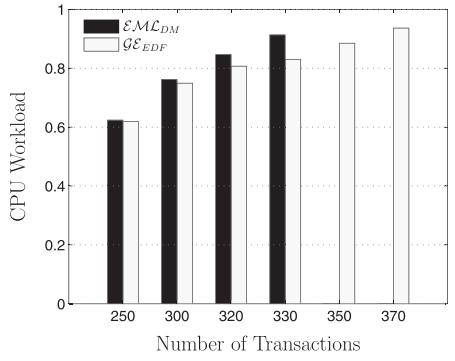

Fig. 5. $\mathcal{EML}_{DM}$ versus $\mathcal{GE}_{EDF}$ with $\mathcal{V}_i$ in $[2,000, 14,000]$ and C_i in $[10, 20]$.

As can be seen from Figs. 4, 5, and 6, $\mathcal{GE}_{EDF}$ consistently outperforms $\mathcal{EML}_{DM}$ on the resulted workload among all the three experiment settings. With the growth of the number of the transactions (N_T), the gap between them is also increasing. Moreover, when N_T exceeding a certain number (different in each setting), $\mathcal{EML}_{DM}$ eventually fails to derive a solution, while $\mathcal{GE}_{EDF}$ can still get one. These experiment results confirmed our analysis in Section 3.2, i.e., EDF yields better workload and feasibility performance than DM with respect to temporal consistency scheduling.

4.2.2 Comparison of Execution Times

The execution times of the six algorithms are shown in Table 6. It can be observed that *HH* and $\mathcal{ML}_{EDF}$ are efficient to derive their solutions. With the growth of transaction set size, the execution times of these three approaches are consistently less than 0.01 s. The reason is that *HH* and $\mathcal{ML}_{EDF}$ are linear-time algorithms. $\mathcal{ML}_{DM}$ has a pseudo-polynomial time complexity in that it requires to solve an iterative equation when determining deadline for each transaction, but it also runs quickly (as can be observed, the computation time is consistently less than 0.01 s), due to that it only takes $\mathcal{O}(\frac{\mathcal{V}_{max}}{2} \cdot n^2)$ time to derive a solution. $\mathcal{EML}_{DM}$ also has pseudopolynomial time complexity since it requires to determine a transaction's deadline by finding out the longest response time which occurs during a *level-i* busy period, but it also runs quickly due to that the number of releases that need be checked is bounded by the validity interval length. Similar to $\mathcal{EML}_{DM}$, $\mathcal{GE}_{EDF}$ has a pseudo-polynomial time complexity, but its actual execution time is determined by the transaction set size. When a solution can be found in Phase 1, $\mathcal{GE}_{EDF}$ can run in linear time, while it takes slightly longer to obtain a solution by Phase 2, as

TABLE 6
Execution Time Comparison

# of Trans.	Execution time (s)						
	HH	$\mathcal{ML}_{EDF}$	$\mathcal{ML}_{DM}$	$\mathcal{EML}_{DM}$	$\mathcal{HS}_{EDF}$	$\mathcal{GE}_{EDF}$	
						I	II
50	< 0.01	< 0.01	< 0.01	0.016	74.53	< 0.01	—
100	< 0.01	< 0.01	< 0.01	0.016	762.53	< 0.01	—
150	< 0.01	< 0.01	< 0.01	0.046	2284.98	< 0.01	—
200	< 0.01	< 0.01	< 0.01	0.046	5383.07	< 0.01	0.016
250	< 0.01	< 0.01	< 0.01	0.075	11943.45	< 0.01	0.031
300	< 0.01	< 0.01	< 0.01	0.091	18726.43	< 0.01	0.062

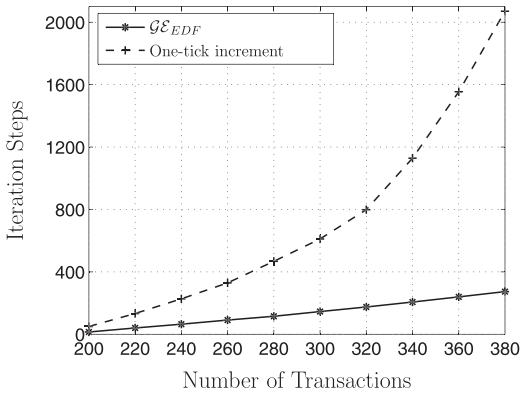

Fig. 7. Iteration steps comparison.

shown in Table 6. Nevertheless, it can finish its execution in a few milliseconds due to the small overhead in its schedulability test.

To illustrate the performance of Theorems 7 and 9 on reducing the overhead of schedulability test, we also conduct the iteration steps comparison between $\mathcal{GE}_{EDF}$ and the one-tick increment scheme, as shown in Fig. 7. In this set of experiment, the range of validity interval length of x_i is set to $[2{,}000, 14{,}000]$, while the execution time of update transactions varies uniformly between 5 and 18 ms. For each point plotted, we have conducted 1,000 runs and take the average. It can be observed that $\mathcal{GE}_{EDF}$ takes significantly fewer iterations than the one-tick increment scheme. With the growing number of transactions, the gap between these two schemes becomes larger. This is because when N_T is small, most of the transactions can be skipped without checking due to $D_i = \sum_{j=1}^{i} C_j$ in both approaches. But with N_T increasing, a significantly large number of iterations are required to be checked in the one-tick increment scheme. Therefore, $\mathcal{GE}_{EDF}$ is efficient when transaction set size scales to 300.

Compared with the other five schemes, $\mathcal{HS}_{EDF}$ takes significantly longer time to obtain a solution, and the execution time increases exponentially when the number of transactions scales up. When $N_T = 50$, the execution time of $\mathcal{HS}_{EDF}$ is about 1 minute. When N_T increases to 300, its execution time increases to almost 5.2 hours. $\mathcal{HS}_{EDF}$ takes much longer time to obtain a solution because it involves solving the 0-1 *knapsack* problem iteratively.

In summary, we revealed the following three observations from our experimental results:

1. $\mathcal{GE}_{EDF}$ and $\mathcal{HS}_{EDF}$ have better schedulability than other approaches (all DM-based ones, and $\mathcal{ML}_{EDF}$), which fail to derive a solution in some settings.
2. $\mathcal{GE}_{EDF}$ consistently outperforms $\mathcal{HS}_{EDF}$ in terms of the resulted processor workload in the whole range of parameter setting.
3. The execution time of $\mathcal{HS}_{EDF}$ increases exponentially when the number of temporal data objects scales up, which makes it less practical for large applications. In contrast, $\mathcal{GE}_{EDF}$ can derive a solution more efficiently, which makes it more attractive to real applications, e.g., air traffic control, stock trading, and vehicular systems, and so on.

5 Related Work

There has been a lot of work on RTDBSs for maintaining real-time data freshness [6], [8], [13], [14], [15], [16], [18], [19], [23], [24], [26], [33]. Song and Liu [26] studied the performance of two well-known concurrency control algorithms, two-phase locking and optimistic, in maintaining temporal consistency of shared data in a hard real-time systems. Kuo and Mok [18] investigated real-time data semantics and proposed a class of real-time access protocol called similarity stack protocol. The tradeoff between data consistency and system workload is exploited in [11], where similarity-based principles are combined with the *HH* scheme to reduce workload by skipping the execution of task instances. Gustafsson and Hansson [8] focused on maintaining data freshness in soft real-time embedded systems and the target application is vehicular systems; an on-demand scheduling algorithm (ODDFT) is proposed for guaranteeing the freshness of base and derived data. Gustafsson and Hansson [7] proposed an algorithm (ODTB) for updating data items that can skip unnecessary updates allowing for better CPU utilization. Lundberg [21] studied the age-constraint problem which is different from our temporal consistency problem. All the work mentioned above assumes the deadlines and periods of update transactions are given, hence gives no answer to the period and deadline assignment problem for maintaining temporal consistency.

The *More-Less* scheme is first proposed in [30] to solve the period and deadline assignment problem with *DM* scheduling, a *fixed priority* scheduling algorithm. While *More-Less* is based on periodic task model, the *DS-FP* proposed in [29] follows a sporadic task model. The *DS-FP* reduces processor workload by adaptively adjusting the separation of two consecutive instances of update transactions while satisfying the validity constraint. Han et al. [9] addressed how to improve the schedulability test condition of *DS-FP*; a necessary and sufficient schedulability condition for *DS-FP*, along with a new schedulability test algorithm, is proposed. As recognized in [9], the scheduling overhead of *DS-FP* is much higher than the periodic scheduling approaches. In this paper, we only focus on investigating periodic scheduling approaches. Jha et al. [13] investigated how to maintain the mutual temporal consistency of real-time data objects. Han et al. [10] studied the problem of how to maintain the temporal validity of real-time data in the presence of mode changes in flexible real-time systems. The authors propose to use different scheduling policies in different modes and introduce two algorithms to search for proper switch points. Very recently, the period and deadline assignment problem for real-time update transactions scheduled under EDF is addressed in [31], in which three algorithms are proposed, as described in Section 1.

In [15], Kang et al. addressed soft real-time data services and present several approaches. Based on the notion of backlog, the authors develop two efficient approaches for fine-grained admission control, built on linear control theory and fuzzy logic control theory, respectively, which are shown to closely support the desired average/transient data service delay. Zhou and Kang [33] addressed how to

support real-time data services in data-intensive real-time applications and proposed a fine-grained deadline assignment scheme that takes the database schema and estimated transaction sizes into account to derive feasible deadlines for real-time data service requests. While the study in [15] and [33] concentrated on user data services request (or user/triggered transactions) and soft real-time data services, we focus on how to assign period and deadline to sensor update transactions, and moreover, we address firm real-time data service. Therefore, our work is complementary to theirs.

As one of the important approach to dealing with overloads in real-time systems, task period adaptations have received considerable attention. Many previous work can be found on the management of overloads in real-time systems based on task period adjustments [17]. But most of them assume that only task periods can change. In [25], task deadlines vary with time, but the tasks do not have periods. The only work that allows task periods and deadlines to change together is presented in [5], but it addresses the case where task deadlines are no larger than their corresponding periods. Note that our work is different from all the work mentioned above which focus on dealing with system overload, since we derive periods and deadlines based on temporal consistency requirements. Balbastre et al.[1] and Hoang et al. [12] have, respectively, addressed the problem of how to find the minimum deadline for each task under EDF scheduling, given that task deadline and period are not related. Our work is different from theirs in that the sum of each transaction's period and deadline is bounded by the validity interval length.

6 Conclusions

Maintaining temporal consistency of real-time data is important in RTDBS. In this work, we have addressed the period and deadline assignment problem for update transactions scheduled by EDF. We proposed a general and efficient two-phase algorithm $\mathcal{GE}_{EDF}$. Our aim of $\mathcal{GE}_{EDF}$ is to produce a feasible schedule while minimizing the CPU workload at the same time, in a time-efficient manner. The first phase of $\mathcal{GE}_{EDF}$ finds a solution in linear time, while the second phase derives a solution by adjusting an $\mathcal{ML}_{DM}$ solution without jeopardizing schedulability. If $\mathcal{ML}_{DM}$ cannot find a solution, the second phase first employs $\mathcal{ML}_{DM}$ to determine a schedulable subset, and then adds one transaction at each step to search for a feasible solution. For schedulability check, we introduced four theorems to reduce the search space, which significantly improve the efficiency of the algorithm. We have conducted intensive experiments with randomly generated transaction sets to evaluate the performance of the proposed techniques. Our experimental results demonstrate that $\mathcal{GE}_{EDF}$ outperforms *HH*, $\mathcal{ML}_{EDF}$, $\mathcal{ML}_{DM}$, and $\mathcal{EML}_{DM}$ in terms of CPU workload and schedulability. Further, our results also show that $\mathcal{GE}_{EDF}$ is much more time efficient than $\mathcal{HS}_{EDF}$. On the whole, due to its effectiveness, $\mathcal{GE}_{EDF}$ can be applied in many practical real-time applications.

For future work, we intend to investigate how to order the transaction set so that a general optimal solution can be achieved. Also, as multiprocessor systems have become a leading trend in embedded real-time applications, we plan to extend our current solutions to multiprocessor platforms.

Acknowledgments

This research was supported, in part, by the National Science Foundation of China (No. 61173049), the Research Fund for the Doctoral Program of the Ministry of Education of China (No. 20090142110023), and a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 115407). The work of LihChyun Shu was partially supported by NSC grant (No. 100-2221-E-006-159). This work was partially done while Jianjun Li was at City University of Hong Kong. The author Jianjun Li thanks Dr. Jian-Jia Chen from Karlsruhe Institute of Technology, Germany, for discussions on the failure condition of $\mathcal{ML}_{DM}$. The authors would also like to thank anonymous reviewers for their comments on improving the quality of this paper.

References

[1] P. Balbastre, I. Ripoll, and A. Crespo, "Minimum Deadline Calculation for Periodic Real-Time Tasks in Dynamic Priority Systems," *IEEE Trans. Computers,* vol. 57, no. 1, pp. 96-109, Jan. 2008.

[2] S. Baruah, L. Rosier, and R. Howell, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor," *Real-Time Systems,* vol. 2, no. 4, pp. 301-324, 1990.

[3] A. Burns and R. Davis, "Choosing Task Periods to Minimise System Utilisation in Time Triggered Systems," *Information Processing Letters,* vol. 58, no. 5, pp. 223-229, 1996.

[4] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Springer, 2005.

[5] T. Chantem, X. Wang, M. Lemmon, and X. Hu, "Period and Deadline Selection for Schedulability in Real-Time Systems," *Proc. Euromicro Conf. Real-Time Systems (ECRTS '08),* pp. 168-177, 2008.

[6] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes," *Proc. IEEE Real-Time Systems Symp.,* pp. 192-203, 1994.

[7] T. Gustafsson and J. Hansson, "Data Management in Real-Time Systems: A Case of on-Demand Updates in Vehicle Control Systems," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp.,* pp. 182-191, 2004.

[8] T. Gustafsson and J. Hansson, "Dynamic on-Demand Updating of Data in Real-Time Database Systems," *Proc. ACM Symp. Applied Computing,* pp. 846-853, 2004.

[9] S. Han, D. Chen, M. Xiong, and A. Mok, "A Schedulability Analysis of Deferrable Scheduling Using Patterns," *Proc. Euromicro Conf. Real-Time Systems (ECRTS),* pp. 47-56, 2008.

[10] S. Han, D. Chen, M. Xiong, and A. Mok, "Online Scheduling Switch for Maintaining Data Freshness in Flexible Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.,* pp. 115-124, 2009.

[11] S. Ho, T. Kuo, and A. Mok, "Similarity-Based Load Adjustment for Real-Time Data-Intensive Applications," *Proc. IEEE Real-Time Systems Symp.,* pp. 144-154, 1997.

[12] H. Hoang, G. Buttazzo, M. Jonsson, and S. Karlsson, "Computing the Minimum EDF Feasible Deadline in Periodic Systems," *Proc. Int'l Conf. Embedded and Real-Time Computing Systems and Applications,* pp. 125-134, 2006.

[13] A. Jha, M. Xiong, and K. Ramamritham, "Mutual Consistency in Real-Time Databases," *Proc. IEEE Real-Time Systems Symp.,* pp. 335-343, 2006.

[14] K. Kang, S. Son, J. Stankovic, and T. Abdelzaher, "A QoS-Sensitive Approach for Timeliness and Freshness Guarantees in Real-Time Databases," *Proc. Euromicro Conf. Real-Time Systems (ECRTS),* 2002.

[15] K. Kang, Y. Zhou, and J. Oh, "Estimating and Enhancing Real-Time Data Service Delays: Control Theoretic Approaches," *IEEE Trans. Knowledge and Data Eng.,* vol. 23, no. 4, pp. 554-567, Apr. 2011.

[16] Y. Kim and S. Son, "Predictability and Consistency in Real-Time Database Systems," *Advances in Real-Time Systems,* pp. 509-531, Prentice-Hall, 1993.

[17] T. Kuo and A. Mok, "Load Adjustment in Adaptive Real-Time Systems," *Proc. IEEE Real-Time Systems Symp.,* pp. 160-171, 1991.

[18] T. Kuo and A. Mok, "Real-Time Data Semantics and Similarity-Based Concurrency Control," *IEEE Trans. Computers,* vol. 49, no. 11, pp. 1241-1254, Nov. 2000.
[19] K. Lam, M. Xiong, B. Liang, and Y. Guo, "Statistical Quality of Service Guarantee for Temporal Consistency of Real-Time Data Objects," *Proc. IEEE Real-Time Systems Symp.,* 2004.
[20] D. Locke, "Real-Time Databases: Real-World Requirements," *Real-Time Databases Systems: Issues and Applications,* pp. 83-92, Kluwer, 1997.
[21] L. Lundberg, "Utilization Based Schedulability Bounds for Age Constraint Process Sets in Real-Time Systems," *Real-Time Systems,* vol. 23, no. 3, pp. 273-295, 2002.
[22] K. Ramamritham, "Real-Time Databases," *Distributed and Parallel Databases,* vol. 1, no. 2, pp. 199-226, 1993.
[23] K. Ramamritham, "Where Do Time Constraints Come From? Where Do They Go?" *J. Database Management,* vol. 7, pp. 4-11, 1996.
[24] K. Ramamritham, S. Son, and L. Dipippo, "Real-Time Databases and Data Services," *Real-Time Systems,* vol. 28, no. 2, pp. 179-215, 2004.
[25] C. Shih and J. Liu, "State-Dependent Deadline Scheduling," *Proc. IEEE Real-Time Systems Symp.,* pp. 3-14, 2002.
[26] X. Song and J. Liu, "Maintaining Temporal Consistency: Pessimistic vs. Optimistic Concurrency Control," *IEEE Trans. Knowledge and Data Eng.,* vol. 7, no. 5, pp. 786-796, Oct. 1995.
[27] M. Spuri, "Analysis of Deadline Scheduled Real-Time Systems," Technical Report 2772, INRIA, 1996.
[28] S. Vestal, "Real-Time Sampled Signal Flows through Asynchronous Distributed Systems," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp.,* pp. 170-179, 2005.
[29] M. Xiong, S. Han, K. Lam, and D. Chen, "Deferrable Scheduling for Maintaining Real-Time Data Freshness: Algorithms, Analysis, and Results," *IEEE Trans. Computers,* vol. 57, no. 7, pp. 952-964, July 2008.
[30] M. Xiong and K. Ramamritham, "Deriving Deadlines and Periods for Real-Time Update Transactions," *IEEE Trans. Computers,* vol. 53, no. 5, pp. 567-583, May 2004.
[31] M. Xiong, Q. Wang, and K. Ramamritham, "On Earliest Deadline First Scheduling for Temporal Consistency Maintenance," *Real-Time Systems,* vol. 40, no. 2, pp. 208-237, 2008.
[32] F. Zhang and A. Burns, "Schedulability Analysis for Real-Time Systems with EDF Scheduling," *IEEE Trans. Computers,* vol. 58, no. 9, pp. 1250-1258, Sept. 2009.
[33] Y. Zhou and K. Kang, "Deadline Assignment and Tardiness Control for Real-Time Data Services," *Proc. Euromicro Conf. Real-Time Systems (ECRTS),* 2010.

Jianjun Li is currently working toward the PhD degree at the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interests include real-time systems, real-time databases, and energy-aware real-time scheduling.

Ming Xiong received the BS degree in computer science and engineering from Xian Jiaotong University, the MS degree in computer science from Sichuan University, China, and the PhD degree in computer science from the University of Massachusetts, Amherst. His research interests include real-time systems, database systems, and mobile computing. From 2000 to 2009, he was a member of the Technical Staff with Bell Laboratories Research, Lucent Technologies, Murray Hill, New Jersey. He is currently a member of the Technical Staff at Google, Inc. He is a member of the IEEE and ACM.

Victor C.S. Lee received the PhD degree in computer science from the City University of Hong Kong in 1997. He is currently an assistant professor in the Department of Computer Science, City University of Hong Kong. His research interests include real-time databases, data management in mobile and wireless computing and performance evaluation. He is a member of the ACM, IEEE and IEEE Computer Society. He has been the chairman of the IEEE, Hong Kong Section, Computer Chapter in 2006-2007.

LihChyun Shu received the PhD degree in computer sciences from Purdue University, West Lafayette, in 1994. He is a professor at the College of Management, National Cheng Kung University, Tainan, Taiwan, ROC. From August 2011, he also serves as the dean at the College of Information and Engineering, Chang Jung Christian University, Tainan, Taiwan, ROC. His research interests include real-time systems, data stream and process mining, and location-based query processing. He is a member of the IEEE and IEEE Computer Society.

Guohui Li received the PhD degree in computer science from the Huazhong University of Science and Technology (HUST), China, in 1999. He was promoted to a full professor in 2004, and currently acts as the vice dean at the School of Computer Science and Technology, HUST. His research interests mainly include real-time systems, mobile computing, and advanced data management. He serves as the corresponding author of this article.