Deferrable Scheduling for Maintaining Real-Time Data Freshness: Algorithms, Analysis, and Results

Ming Xiong
Bell Labs, Alcatel-Lucent
xiong@research.bell-labs.com

Song Han *
The University of Texas at Austin
shan@cs.utexas.edu

Kam-Yiu Lam
City University of Hong Kong
cskylam@cityu.edu.hk

Deji Chen
Emerson Process Management
Deji.Chen@EmersonProcess.com

Abstract

The periodic update transaction model has been used to maintain freshness (or temporal validity) of real-time data. Period and deadline assignment has been the main focus in the past studies such as the More-Less scheme [25] in which update transactions are guaranteed by the Deadline Monotonic scheduling algorithm [16] to complete by their deadlines. In this article, we propose a deferrable scheduling *algorithm for fixed priority transactions – a novel approach for minimizing update workload while maintaining the temporal validity of real-time data. In contrast to prior work on maintaining data freshness periodically, update transactions follow an aperiodic task model in the deferrable scheduling algorithm. The deferrable scheduling algorithm exploits the semantics of temporal validity constraint of real-time data by judiciously deferring the sampling times of update transaction jobs as late as possible. We present a theoretical estimation of its processor utilization, and a sufficient condition for its schedulability. Our experimental results verify the theoretical estimation of the processor utilization. We demonstrate through the experiments that the deferrable scheduling algorithm is an effective approach, and it significantly outperforms the More-Less scheme in terms of reducing processor workload.*

Keywords - *Deferrable scheduling, real-time databases, temporal validity, fixed priority scheduling*

1 INTRODUCTION

Real-time and embedded systems are applied in many application domains that require timely processing of massive amount of real-time data. Examples of real-time data include sensor data in sensor networks, positions of aircrafts in air traffic control systems [14], and vehicle velocity in adaptive cruise control applications [6]. Such real-time data are typically managed in a real-time database system (RTDBS). Those data values are used to model the current status of entities in a system environment. However, real-time data are different from traditional data in that they have time semantics in which sampled values are valid only for a certain time interval [19, 18, 23]. The concept of *temporal validity* is used to define the correctness of real-time data [19]. A real-time data object is *fresh* (or *temporally valid*) if its value truly reflects the current status of the corresponding entity in the system environment. Each real-time data object is associated with a *validity interval* as the lifespan of the current data value defined based on the dynamic properties of the data object. A new data value needs to be installed into the database before the validity interval of the old value expires, i.e., the old one becomes temporally invalid. Otherwise, the RTDBS cannot detect and respond to environmental changes in a timely manner. In recent years, there has been tremendous amount of work devoted to this area [5, 1, 12, 14, 30, 19, 20, 21, 22, 26, 11, 25, 8].

To maintain temporal validity, *sensor update transactions*, which capture the latest status of the entities in the system environment, are generated to refresh the values of the real-time data periodically [19, 14, 25]. A sensor update transaction has an infinite number of periodic jobs, which have fixed length periods and relative deadlines. The update problem for periodic update transactions consists of two parts [25]: *(1) the determination of the sampling periods and deadlines of update transactions; and (2) the scheduling of update transactions*. Prior work has proposed two approaches for minimizing the update workload while maintaining real-time data freshness. As explained in [19, 14], a simple method to maintain the temporal validity of real-time data is to use the *Half-Half (HH)* scheme in which the update period for a real-time data object is set to be half of

*This work was partially done while the co-author was at City University of Hong Kong.

the validity interval of the object. To further reduce the update workload, the *More-Less (ML)* scheme is proposed and studied in [2, 25].

This article presents *Deferrable Scheduling* for *Fixed Priority* transactions (*DS-FP*), a novel algorithm for maintaining real-time data freshness, with the objective being to minimize the update workload [27, 28]. We study the problem of data freshness maintenance for firm real-time update transactions in a single processor RTDBS. Distinct from the past work of *HH* and *ML*, which have a fixed period and relative deadline for each transaction, *DS-FP* adopts an *aperiodic* task model. In contrast to *ML*, in which a relative deadline is always equivalent to the worst-case response time of a transaction, *DS-FP* dynamically assigns relative deadlines to transaction jobs by deferring the sampling time of a transaction job as much as possible while still guaranteeing the temporal validity of real-time data. The deferral of a job's sampling time results in a shorter relative deadline than its worst-case response time, which in turn increases the separation of two consecutive jobs. Thus, the deferral of sampling time lends itself to a reduced processor workload produced by update transactions. We prove that *DS-FP* outperforms *ML* in terms of schedulability and present a sufficient condition for the schedulability of a set of transactions under *DS-FP*. We also analyze the average processor utilization under *DS-FP*. Our experimental study of *DS-FP* demonstrates that it is an effective algorithm for reducing the workload of real-time update transactions. It also verifies the accuracy of our theoretical estimation of average processor utilization under *DS-FP*, and demonstrates the effectiveness of the *DS-FP* algorithms.

The rest of the article is organized as follows: Section 2 reviews the existing approaches for real-time data freshness maintenance. In Section 3, we propose the Deferrable Scheduling algorithm for Fixed Priority transactions (*DS-FP*). Our detailed discussion on *DS-FP* includes an analysis of its schedulability and non-optimality, as well as an estimation of its average processor utilization. Section 4 presents the performance studies and Section 5 briefly describes the related work. Finally, we conclude our study in Section 6 and present open questions for *DS-FP*.

2 BACKGROUND: DATA FRESHNESS MAINTENANCE

Real-time data, whose state may become invalid with the passage of time, need to be refreshed by sensor update transactions generated by intelligent sensors that sample the values of real world entities. To monitor the states of entities faithfully, real-time data must be refreshed before they become invalid. The actual length of the temporal validity interval of a real-time data object is application-dependent. For example, real-time data with validity interval requirements are discussed in [19, 20, 18]. One of the important design goals of RTDBSs is to guarantee that real-time data remain fresh, i.e., they are always valid.

2.1 Temporal Validity for Data Freshness

As real-time data values change continuously with time, the correctness of a real-time data object X_i depends on the difference between the real-time status $S(E_i)$ of the real world entity E_i and the current sampling value $Val(X_i)$ of X_i.

Definition 2.1: A real-time data object X_i at time t is temporally valid (or temporally consistent) if, for its update job $J_{i,j}$ finished last before t, the sampling time $r_{i,j}$ plus the validity interval length (or validity length for short) $\mathcal{V}_i$ of the data object is not less than t, i.e., $r_{i,j} + \mathcal{V}_i \geq t$ [21, 19, 1]. □

A data value for real-time data object X_i sampled at any time t will be valid for $\mathcal{V}_i$ following that t up to $(t + \mathcal{V}_i)$. Next, we review existing approaches that adopt a periodic task model for sensor update transactions.

2.2 Half-Half and More-Less

In this section, traditional approaches for maintaining temporal validity, namely the *Half-Half* (*HH*) and *More-Less* (*ML*) approaches are reviewed.

In this article, $\mathcal{T} = \{\tau_i\}_{i=1}^m$ refers to a set of periodic update transactions $\{\tau_1, \tau_2, .., \tau_m\}$ and $\mathcal{X} = \{X_i\}_{i=1}^m$ refers to a set of real-time data objects. We assume that τ_i has higher priority than τ_j for $i < j$ unless specified otherwise. All real-time data objects are assumed to be kept in main memory. Associated with X_i $(1 \leq i \leq m)$ is a validity interval of length $\mathcal{V}_i$: transaction τ_i $(1 \leq i \leq m)$ updates the corresponding data object X_i. Because each update transaction updates a different data object, no concurrency control is considered for update transactions. We assume that a sensor always samples the value of a real-time data object at the beginning of its period, and the system is *synchronous* (i.e., all the first jobs of update transactions are initiated at the same time) unless stated otherwise. For convenience, let $d_{i,j}$, $f_{i,j}$ and $r_{i,j}$ denote the absolute deadline, completion (finishing) time, and sampling (release) time of job $J_{i,j}$ of τ_i, respectively. We also assume that jitter between sampling time and release time of a job is zero for convenience of presentation (readers are referred to Section 3.3 for how jitters can be handled). Formal definitions of the frequently used symbols are given in Table 1. Deadlines of update transactions are firm deadlines. The goal of *Half-Half* and *More-Less*, which adopt a *periodic* task model, is to determine period P_i and relative deadline D_i so that all the update transactions are schedulable and the CPU workload resulting from periodic update transactions is minimized.

Both *HH* and *ML* assume a simple execution semantics for periodic transactions: a transaction must be executed

Symbol	Definition
X_i	Real-time data object i
τ_i	Update transaction updating X_i
$J_{i,j}$	The $(j+1)^{th}$ job of τ_i ($i = 1, .., m,\ j = 0, 1, 2, ..$)
$R_{i,j}$	Response time of $J_{i,j}$
C_i	Computation time of transaction τ_i
$\mathcal{V}_i$	Validity (interval) length of X_i
$f_{i,j}$	Finishing (completion) time of $J_{i,j}$
$r_{i,j}$	Release (Sampling) time of $J_{i,j}$
$d_{i,j}$	Absolute deadline of $J_{i,j}$
P_i	Period of transaction τ_i in *ML*
D_i	Relative deadline of transaction τ_i in *ML*
$P_{i,j}$	Separation of jobs (i.e., $r_{i,j+1} - r_{i,j}$) in *DS-FP*
$D_{i,j}$	Relative deadline of $J_{i,j}$ in *DS-FP*
$\overline{P}_i$	Average period of transaction τ_i in *DS-FP*
$\overline{D}_i$	Average relative deadline of transaction τ_i in *DS-FP*
$\overline{U}_{DS}$	Average processor utilization in *DS-FP*
$\Theta_i(a, b)$	Total cumulative processor demands from higher-priority transactions received by τ_i in interval $[a, b)$

Table 1. Symbols and definitions

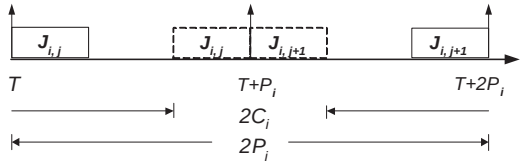


Figure 1. Extreme execution cases of $J_{i,j}$ and $J_{i,j+1}$

once every period. However, there is no guarantee on when a job of a periodic transaction is actually executed within a period. Throughout this article we assume the scheduling algorithms are preemptive and ignore all preemption overhead. For convenience, we use terms transaction and task interchangeably.

Half-Half: In *HH*, the period and relative deadline of an update transaction are each typically set to be one-half of the data validity length [19, 14]. In Figure 1, the farthest distance of two consecutive jobs of τ_i (based on the sampling time $r_{i,j}$ of job $J_{i,j}$ and the deadline $d_{i,j+1}$ of its next job) is $2P_i$. If $2P_i \leq \mathcal{V}_i$, then the validity of real-time object X_i is guaranteed as long as jobs of τ_i meet their deadlines. Unfortunately, this approach incurs an unnecessarily high CPU workload of update transactions in the RTDBSs compared to *More-Less*.

More-Less: Consider the worst-case response time for any job of a periodic transaction τ_i where the response time is the difference between the transaction initiation time $(I_i + KP_i)$ and the transaction completion time where I_i is the offset within the period and K is a natural number.

Lemma 2.1: For a set of periodic transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ $(D_i \leq P_i)$ with transaction initiation time $(I_i + KP_i)$ $(K = 0, 1, 2, ...)$, the worst-case response time for any job of τ_i occurs for the first job of τ_i when $I_1 = I_2 = ... = I_m = 0$. [16] □

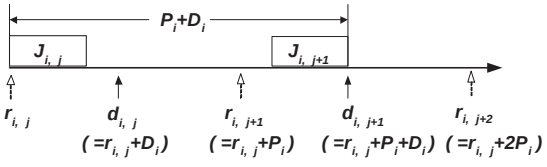


Figure 2. Illustration of *More-Less* scheme

For $I_i = 0$ $(1 \leq i \leq m)$, the transactions are *synchronous*. A time instant after which a transaction has the worst-case response time is called a *critical instant*, e.g., time 0 is a critical instant for all the transactions if those transactions are *synchronous*.

To minimize the update workload and guarantee temporal validity, *ML* uses *Deadline Monotonic* (*DM*) [16] to schedule periodic update transactions [2, 25]. There are three constraints to follow for τ_i $(1 \leq i \leq m)$:

- *Validity constraint*: the sum of the period and relative deadline of transaction τ_i is always less than or equal to $\mathcal{V}_i$, i.e., $P_i + D_i \leq \mathcal{V}_i$, as shown in Figure 2.
- *Deadline constraint*: the period of an update transaction is assigned to be more than half of the validity length of the object to be updated, while its corresponding relative deadline is less than half of the validity length of the same object. For τ_i to be schedulable, D_i must be greater than or equal to C_i, the worst-case execution time of τ_i, i.e., $C_i \leq D_i \leq P_i$.
- *Schedulability constraint*: for a given set of update transactions, the *Deadline Monotonic* scheduling algorithm [16] is used to schedule the transactions. Consequently, $\sum_{j=1}^{i}(\lceil \frac{D_i}{P_j} \rceil \cdot C_j) \leq D_i$ $(1 \leq i \leq m)$ if τ_j has higher priority than τ_i for $i > j$.

ML assigns priorities to transactions based on *Shortest Validity First* (SVF), i.e., in the *inverse* order of validity length and ties are resolved in favor of transactions with less slack (i.e., $\mathcal{V}_i - C_i$ for τ_i). It assigns deadlines and periods to τ_i as follows:

$$D_i = f_{i,0}^{ml} - r_{i,0}^{ml}, \quad (1)$$

$$P_i = \mathcal{V}_i - D_i, \quad (2)$$

where $f_{i,0}^{ml}$ and $r_{i,0}^{ml}$ are finishing and sampling times of the first job of τ_i under *ML*, respectively. Note that in a synchronous system, $r_{i,0}^{ml} = 0$ and the first job's response time is the worst-case response time in *ML*. In this article, superscript *ml* is used to distinguish the finishing and sampling times in *ML* from those in *DS-FP*.

3 DEFERRABLE SCHEDULING

All schedulers discussed in this article are *work-conserving* for released jobs. In other words, the scheduler never idles the processor while there is a job awaiting

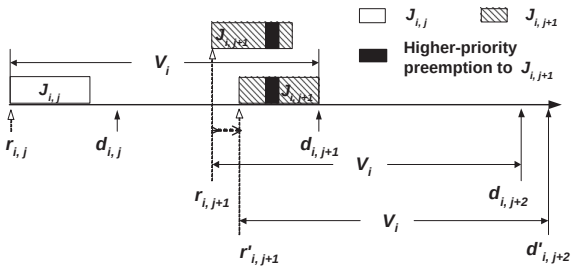


Figure 3. Illustration of *DS-FP* scheduling ($r_{i,j+1}$ is shifted to $r'_{i,j+1}$)

execution (i.e., after it is released). Next, we introduce the *Deferrable Scheduling algorithm for Fixed Priority* transactions (*DS-FP*). Section 3.1 presents the intuition of the algorithm, and Section 3.2 describes the details of the algorithm. Section 3.3 compares it with *ML*. Section 3.4 provides an estimation of *DS-FP*'s average processor utilization. Section 3.5 discusses whether the algorithm is optimal.

3.1 Intuition of *DS-FP*

In *ML*, D_i is determined by the first job's response time, which is the *worst-case* response time of all jobs of τ_i. Thus, *ML* is pessimistic on the deadline and period assignment in the sense that it uses a periodic task model that has a fixed period and relative deadline for each task, and the relative deadline is equivalent to the worst-case response time. It should be noted that the *validity constraint* can always be satisfied as long as $P_i + D_i \leq \mathcal{V}_i$. However, the processor workload is minimized only if $P_i + D_i = \mathcal{V}_i$. Otherwise, P_i can always be increased to reduce processor workload as long as $P_i + D_i < \mathcal{V}_i$. Given release time $r_{i,j}$ of job $J_{i,j}$ and deadline $d_{i,j+1}$ of job $J_{i,j+1}$ $(j \geq 0)$,

$$d_{i,j+1} \leq r_{i,j} + \mathcal{V}_i \tag{3}$$

guarantees that the *validity constraint* can be satisfied, as depicted in Figure 3. Correspondingly, the following equation follows directly from Eq. 3.

$$(r_{i,j+1} - r_{i,j}) + (d_{i,j+1} - r_{i,j+1}) \leq \mathcal{V}_i. \tag{4}$$

If $r_{i,j+1}$ is shifted onward to $r'_{i,j+1}$ along the time line in Figure 3, it does not violate Eq. 4 and $J_{i,j+1}$ can still be completed by its deadline. This shift can be achieved, e.g., in the *ML* schedule, if preemption to $J_{i,j+1}$ from higher-priority transactions in $[r_{i,j+1}, d_{i,j+1}]$ is less than the worst-case preemption to the first job of τ_i. Thus, temporal validity can still be guaranteed as long as $J_{i,j+1}$ is completed by its deadline $d_{i,j+1}$.

The intention of *DS-FP* is to defer the sampling time, $r_{i,j+1}$, of $J_{i,j}$'s subsequent job as late as possible while still guaranteeing the *validity constraint*. Note that the sampling time of a job is also its release time, i.e., the time that the job is ready to execute, as we assume zero cost for sampling and no arrival jitter for a job for convenience of presentation.

The deferral of job $J_{i,j+1}$'s release time reduces the relative deadline of the job if its absolute deadline is fixed as in Eq. 3. For example, although $r_{i,j+1}$ is deferred to $r'_{i,j+1}$ in Figure 3, it still has to be completed by its deadline $d_{i,j+1}$ in order to satisfy the *validity constraint* (Eq. 3). Thus its relative deadline, $D_{i,j+1}$, becomes $d_{i,j+1} - r'_{i,j+1}$, which is less than $d_{i,j+1} - r_{i,j+1}$. The deadline of $J_{i,j+1}$'s subsequent job, $J_{i,j+2}$, can be further deferred to $(r'_{i,j+1} + \mathcal{V}_i)$ to satisfy the *validity constraint*. Consequently, the processor utilization for completion of three jobs, $J_{i,j}$, $J_{i,j+1}$, and $J_{i,j+2}$ then becomes $\frac{3C_i}{2\mathcal{V}_i - (d_{i,j+1} - r'_{i,j+1})}$. It is less than the utilization $\frac{3C_i}{2\mathcal{V}_i - (d_{i,j+1} - r_{i,j+1})}$ required for completion of the same amount of work in *ML*.

Definition 3.1: Let $\Theta_i(a, b)$ denote the total *cumulative processor demands* made by all jobs of higher-priority transaction τ_j for $\forall j$ $(1 \leq j \leq i-1)$ during the time interval $[a, b)$ from a schedule $\mathcal{S}$ produced by a fixed priority scheduling algorithm. Then,

$$\Theta_i(a, b) = \sum_{j=1}^{i-1} \theta_j(a, b),$$

where $\theta_j(a, b)$ is the total processor demands made by all jobs of single transaction τ_j during $[a, b)$. □

Next, we discuss how much a job's release time can be deferred. We shall use $r_{i,j+1}$ instead of $r'_{i,j+1}$ to denote the final deferred release time. According to fixed priority scheduling theory, $r_{i,j+1}$ can be derived backwards from its deadline $d_{i,j+1}$ as follows:

$$r_{i,j+1} = d_{i,j+1} - R_{i,j+1}(r_{i,j+1}, d_{i,j+1}); \tag{5}$$

$$R_{i,j+1}(r_{i,j+1}, d_{i,j+1}) = \Theta_i(r_{i,j+1}, d_{i,j+1}) + C_i; \tag{6}$$

where $R_{i,j+1}(r_{i,j+1}, d_{i,j+1})$ denotes the response time of $J_{i,j+1}$ in the time interval $[r_{i,j+1}, d_{i,j+1})$. Note that the schedule of all higher-priority jobs that are released prior to $d_{i,j+1}$ needs to be computed before $\Theta_i(r_{i,j+1}, d_{i,j+1})$ is computed. This computation can be invoked using a recursive process from jobs of lower-priority transactions to higher-priority transactions. Nevertheless, it does not require that a *complete* schedule of all jobs should be constructed off-line before the task set is executed. Indeed, the computation of job deadlines and their corresponding release times is performed on-line while the transactions are being scheduled. We only need to compute the first jobs' response times when system starts. Upon the completion of job $J_{i,j}$, the deadline of its next job, $d_{i,j+1}$, is firstly derived

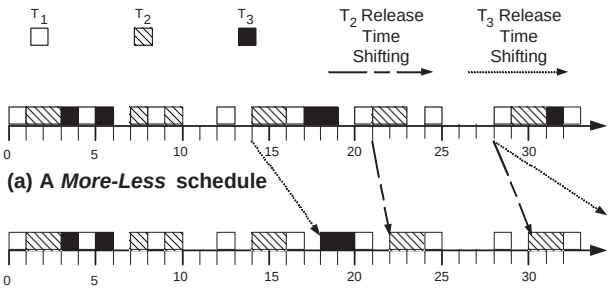


Figure 4. Comparing *ML* and *DS-FP* schedules

from Eq. 3, then the corresponding release time $r_{i,j+1}$ is derived from Eq. 5. If $\Theta_i(r_{i,j+1}, d_{i,j+1})$ cannot be computed due to incomplete schedule information of release times and absolute deadlines from higher-priority transactions, *DS-FP* computes their schedule information on-line until it can gather enough information to derive $r_{i,j+1}$. Job $J_{i,j}$'s *DS-FP* scheduling information (e.g., release time, deadline, bookkeeping information, etc.) can be discarded after it is completed and no lower-priority transactions need its information for deriving their schedules. This process is called *garbage collection* in *DS-FP*.

Let $S_J(t)$ denote the set of jobs of all transactions whose deadlines have been computed by time t. Also let $LSD_i(t)$ denote the latest scheduled deadline of τ_i at t, i.e., maximum of all $d_{i,j}$ for jobs $J_{i,j}$ of τ_i whose deadlines have been computed by t, then

$$LSD_i(t) = \max_{J_{i,j} \in S_J(t)} \{d_{i,j}\} \ (j \geq 0). \tag{7}$$

Given job $J_{k,j}$ whose scheduling information has been computed at time t, and $\forall i \ (i > k)$, if

$$LSD_i(t) \geq d_{k,j}, \tag{8}$$

then the information of $J_{k,j}$ can be garbage collected.

Example 3.1: Suppose that there are three update transactions whose parameters are shown in Table 2. The resulting periods and deadlines in *HH* and *ML* are shown in the same table. Utilizations of *HH* and *ML* are $\mathcal{U}_{ml} \approx 0.68$ and $\mathcal{U}_{hh} = 1.00$, respectively.

Figures 4 (a) and (b) depict the schedules produced by *ML* and *DS-FP*, respectively. It can be observed from both schedules that the release times of transaction jobs $J_{3,1}$, $J_{2,3}$, $J_{2,4}$ are shifted from times $14,\ 21,\ 28$ in *ML* to $18,\ 22,\ 30$ in *DS-FP*, respectively. □

The *DS-FP* algorithm is described in Section 3.2.

3.2 *Deferrable Scheduling* Algorithm

This subsection presents *DS-FP*, a *fixed priority* scheduling algorithm. Transaction priority assignment policy in *DS-FP* is the same as in *ML*, i.e., *Shortest Validity First*. Given an update transaction set $\mathcal{T}$, it is assumed that τ_i has a priority higher than τ_j if $i < j$ as we let $\mathcal{V}_i \leq \mathcal{V}_j$. Algorithm 3.1 presents the *DS-FP* algorithm. For convenience of presentation, garbage collection is omitted in the algorithm. There are two cases for the *DS-FP* algorithm: 1) At system initialization time, Lines 13 to 20 iteratively calculate the first job's response time for τ_i. The first job's deadline is set as its response time (Line 21). 2) Upon completion of τ_i's job $J_{i,k}$ $(1 \leq i \leq m, k \geq 0)$, the deadline of its next job ($J_{i,k+1}$), $d_{i,k+1}$, is derived at Line 27 so that the farthest distance of $J_{i,k}$'s sampling time and $J_{i,k+1}$'s finishing time is bounded by the validity length $\mathcal{V}_i$ (Eq. 3). Finally, the sampling time of $J_{i,k+1}$, $r_{i,k+1}$, is derived backwards from its deadline by accounting for the interferences from higher-priority transactions (Line 29).

i	C_i	$\mathcal{V}_i$	$\frac{C_i}{\mathcal{V}_i}$	ML		*Half-Half*
				P_i	D_i	$P_i(D_i)$
1	1	5	0.2	4	1	2.5
2	2	10	0.2	7	3	5
3	2	20	0.1	14	6	10

Table 2. Parameters and results for Example 3.1

Algorithm 3.1 *DS-FP* **algorithm:**

Input: *A set of update transactions* $\mathcal{T} = \{\tau_i\}_{i=1}^m \ (m \geq 1)$ *with known* $\{C_i\}_{i=1}^m$ *and* $\{\mathcal{V}_i\}_{i=1}^m$.

Output: *Construct a partial schedule* $\mathcal{S}$ *if* $\mathcal{T}$ *is feasible; otherwise, reject.*

```
1  case (system initialization time) :
2  t ← 0; // Initialization
3  // LSD_i – Latest Scheduled Deadline of τ_i's jobs.
4  LSD_i ← 0, ∀i (1 ≤ i ≤ m);
5  ℓ_i ← 0, ∀i (1 ≤ i ≤ m);
6  // ℓ_i is the latest scheduled job of τ_i
7  for i = 1 to m do
8      // Schedule finish time for τ_{i,0}.
9      r_{i,0} ← 0;
10     f_{i,0} ← C_i;
11     // Calculate higher-priority (HP) preemptions.
12     oldHPPreempt ← 0;        // initial HP preemptions
13     hpPreempt ← CalcHPPreempt(i, 0, 0, f_{i,0});
14     while (hpPreempt > oldHPPreempt) do
15         // Accounting for the interferences of HP tasks
16         f_{i,0} ← r_{i,0} + hpPreempt + C_i;
17         if (f_{i,0} > V_i − C_i) then abort endif;
18         oldHPPreempt ← hpPreempt;
19         hpPreempt ← CalcHPPreempt(i, 0, 0, f_{i,0});
20     end
21     d_{i,0} ← f_{i,0};
22 end
23 return;
```

```
25 case (upon completion of J_{i,k}) :
26 // Schedule release time for J_{i,k+1}.
27 d_{i,k+1} ← r_{i,k} + V_i;      // get next deadline for J_{i,k+1}
28 // r_{i,k+1} is also the sampling time for J_{i,k+1}
29 r_{i,k+1} ← ScheduleRT(i, k+1, C_i, d_{i,k+1});
30 return;
```

Algorithm 3.2 ScheduleRT$(i, k, C_i, \mathrm{d}_{i,k})$**:**

Input: $J_{i,k}$ *with* C_i *and* $d_{i,k}$.

Output: $r_{i,k}$.

```
1  oldHPPreempt ← 0;          // initial HP preemptions
2  hpPreempt ← 0;
3  r_{i,k} ← d_{i,k} − C_i;
4  // Calculate HP preemptions backwards from d_{i,k}.
5  hpPreempt ← CalcHPPreempt(i, k, r_{i,k}, d_{i,k});
6  while (hpPreempt > oldHPPreempt) do
7        // Accounting for the interferences of HP tasks
8        r_{i,k} ← d_{i,k} − hpPreempt − C_i;
9        if (r_{i,k} < d_{i,k−1}) then abort endif;
10       oldHPPreempt ← hpPreempt;
11       hpPreempt ← GetHPPreempt(i, k, r_{i,k}, d_{i,k});
12 end
13 return r_{i,k};
```

Algorithm 3.3 CalcHPPreempt(i, k, t_1, t_2)**:**

Input: $J_{i,k}$*, and a time interval* $[t_1, t_2)$.

Output: *Total cumulative processor demands from higher-priority transactions* τ_j $(1 \le j \le i-1)$ *during* $[t_1, t_2)$.

```
1  ℓ_i ← k;          // Record latest scheduled job of τ_i.
2  d_{i,k} ← t_2;
3  LSD_i ← t_2;
4  if (i = 1)
5    then  // No preemptions from higher-priority tasks.
6          return 0;
7  elsif (LSD_{i−1} ≥ LSD_i)
8    then      // Get preemptions from τ_j (∀j, 1 ≤ j < i)
9          // because τ_j's schedule is complete before t_2.
10         return GetHPPreempt(i, k, t_1, LSD_i);
11 endif
12 // build S up to or exceeding t_2 for τ_j (1 ≤ j < i).
13 for j = 1 to i − 1 do
14    while (d_{j,ℓ_j} < LSD_i) do
15       d_{j,ℓ_j+1} ← r_{j,ℓ_j} + V_j;
16       r_{j,ℓ_j+1} ← ScheduleRT(j, ℓ_j + 1, C_j, d_{j,ℓ_j+1});
17       ℓ_j ← ℓ_j + 1;
18       LSD_j ← d_{j,ℓ_j};
19    end
20 end
21 return GetHPPreempt(i, k, t_1, LSD_i);
```

Function *ScheduleRT*$(i,\ k,\ C_i,\ d_{i,k})$ (Algorithm 3.2) calculates the release time $r_{i,k}$ with known computation time C_i and deadline $d_{i,k}$. It starts with release time $r_{i,k} = d_{i,k} - C_i$, then iteratively calculates $\Theta_i(r_{i,k},\ d_{i,k})$, the total cumulative processor demands made by all higher-priority jobs of $J_{i,k}$ during the interval $[r_{i,k},\ d_{i,k})$, and adjusts $r_{i,k}$ by accounting for the interferences from higher-priority transactions (Lines 5 to 12). The computation of $r_{i,k}$ continues until the interferences from higher-priority transactions do not change in an iteration. In particular, Line 9 detects any infeasible schedule. A schedule becomes infeasible under *DS-FP* if $r_{i,k} < d_{i,k-1}$ $(k > 0)$, i.e., release time of $J_{i,k}$ becomes earlier than the deadline of its preceding job $J_{i,k-1}$. Function *GetHPPreempt*$(i,\ k,\ t_1,\ t_2)$ scans the interval $[t_1, t_2)$, adds up total preemptions from τ_j $(\forall j,\ 1 \le j \le i-1)$, and returns $\Theta_i(t_1, t_2)$, the cumulative processor demands of τ_j during $[t_1, t_2)$ from schedule $\mathcal{S}$ that has been built.

Function *CalcHPPreempt*$(i,\ k,\ t_1,\ t_2)$ (Algorithm 3.3) calculates $\Theta_i(t_1,\ t_2)$, the total cumulative processor demands made by all higher-priority jobs of $J_{i,k}$ during the interval $[t_1,\ t_2)$. Line 7 ensures that $(\forall j, 1 \le j < i)$, τ_j's schedule is completely built before time t_2. This is because τ_i's schedule cannot be completely built before t_2 unless the schedules of its higher-priority transactions are complete before t_2. In this case, the function simply returns an amount of higher-priority preemptions for τ_i during $[t_1, t_2)$ by invoking *GetHPPreempt*$(i,\ k,\ t_1,\ t_2)$, which returns $\Theta_i(t_1, t_2)$. If any higher-priority transaction τ_j $(j < i)$ does not have a complete schedule during $[t_1,\ t_2)$, its schedule $\mathcal{S}$ up to or exceeding t_2 is built on the fly (Lines 14 to 19). This enables the computation of $\Theta_i(t_1,\ t_2)$. The latest scheduled deadline of τ_i's job, LSD_i, indicates the latest deadline of τ_i's jobs that have been computed.

The *worst-case* complexity of *ScheduleRT* is $O(m \cdot \mathcal{V}_m^2)$ assuming that $\frac{\mathcal{V}_m}{\mathcal{V}_1}$ is a constant. An important property of *ScheduleRT*$(i,\ k,\ C_i,\ d_{i,k})$ terminating at time $t = d_{i,k}$ is that the latest scheduled deadline of τ_l $(LSD_l(t))$ is no larger than that of τ_j $(LSD_j(t))$ if τ_l does not have a priority higher than τ_j $(l \ge j)$. This is proved in the following lemma.

Lemma 3.1: Given a synchronous update transaction set $\mathcal{T}$ and *ScheduleRT*$(i,\ k,\ C_i,\ t)$ $(1 \le i \le m\ \&\ k \ge 0)$, $LSD_l(t) \le LSD_j(t)$ $(i \ge l \ge j)$ holds when *ScheduleRT*$(i,\ k,\ C_i,\ t)$ terminates at time t.

Proof. This can be proved by contradiction. Suppose that $LSD_l(t) > LSD_j(t)$ $(i \ge l \ge j)$ when *ScheduleRT*$(i,\ k,\ C_i,\ t)$ terminates at t. If $LSD_l(t) < t$, then *CalcHPPreempt*$(i,\ k,\ t_1,\ t_2)$ does not terminate according to Line 14 because $d_{l,\ell_l} < LSD_i(t) = t$. Thus $LSD_l(t) \ge LSD_i(t) = t$. Let $LSD_l(t) = t_2$ in *CalcH-*

	τ_1	τ_2		τ_3	
Job	*ML/DS-FP*	*ML*	*DS-FP*	*ML*	*DS-FP*
0	(0,1)	(0, 3)	(0, 3)	(0, 6)	(0, 6)
1	(4,5)	(7, 10)	(7, 10)	(14, 20)	(18, 20)
2	(8,9)	(14, 17)	(14, 17)	(28, 34)	(35, 38)
3	(12,13)	(21, 24)	(22, 24)	...	...
4	(16,17)	(28, 31)	(30, 32)		
5	(20,21)	(35, 38)	(38, 40)		
6	(24,25)	...	...		
7	(28,29)				
8	(32,33)				
9	(36,37)				

Table 3. Release time and deadline comparison

PPreempt$(l,\ k_l,\ t_1,\ t_2)$, which must be invoked before *ScheduleRT*$(i,\ k,\ C_i,\ t)$ terminates at t. As we assume that $LSD_j(t) < LSD_l(t) = t_2$, similarly *CalcHPPreempt*$(l,\ k_l,\ t_1,\ t_2)$ has not reached the point to terminate according to Line 14. This contradicts the assumption. □

The next example illustrates how the *DS-FP* algorithm works with the transaction set in Example 3.1.

Example 3.2: Table 3 presents the comparison of (release time, deadline) pairs assigned by *ML* and *DS-FP* (Algorithm 3.1) for the jobs of τ_1, τ_2 and τ_3 in Example 3.1. Note that only release times and deadlines before time 40 are depicted in the table. Please also note that τ_1 has same release times and deadlines for all jobs under *ML* and *DS-FP*. However, $J_{2,3}$, $J_{2,4}$, $J_{2,5}$, $J_{3,1}$, and $J_{3,2}$ have different release times and deadlines under *ML* and *DS-FP*. Algorithm 3.1 starts at *system initialization* time. It calculates deadlines for $J_{1,0}$, $J_{2,0}$, $J_{3,0}$. Upon completion of $J_{3,0}$ at time 6, $d_{3,1}$ is set to $r_{3,0} + \mathcal{V}_3 = 20$. Then Algorithm 3.1 invokes *ScheduleRT*$(3, 1, 2, 20)$ at Line 29, which derives $r_{3,1}$. At this moment, Algorithm 3.1 has already calculated the complete schedule up to $d_{3,0}$ (time 6). But the schedule in the interval $(6, 20]$ has only been partially derived. Specifically, only schedule information of $J_{1,0}$, $J_{1,1}$, $J_{1,2}$, $J_{1,3}$ $J_{2,0}$, and $J_{2,1}$ has been derived for τ_1 and τ_2. Algorithm 3.2 (*ScheduleRT*) obtains $r_{3,1} = 20 - 2 = 18$ at Line 3, then invokes *CalcHPPreempt*$(3, 1, 18, 20)$. Algorithm 3.3 (*CalcHPPreempt*) finds that $LSD_2 = 10 < t_2 = 20$, then it jumps to the *for* loop starting at Line 13 to build the complete schedule of τ_1 and τ_2 in the interval $(6, 20]$, where the release times and deadlines for $J_{1,4}$, $J_{1,5}$, $J_{2,2}$, $J_{1,6}$, and $J_{2,3}$ are derived. Thus, higher-priority transactions τ_1 and τ_2 have a complete schedule before time 20. Note that $r_{1,6}$ and $d_{1,6}$ for $J_{1,6}$ are derived when we calculate $r_{2,3}$ and $d_{2,3}$ such that the complete schedule up to time $d_{2,3}$ is built for transactions with priorities higher than τ_2. As $r_{2,2}$ is set to 14 by earlier calculation, $d_{2,3}$ is set to 24. It derives $r_{2,3}$ backwards from $d_{2,3}$ and sets it to 22 because $\Theta_2(22, 24) = 0$. Similarly, $d_{3,1}$ and $r_{3,1}$ are set to 20 and 18, respectively. □

3.3 Comparison of *DS-FP* and *ML*

Note that *ML* is based on the *periodic* task model, while *DS-FP* adopts the *aperiodic* task model. The relative deadline of a transaction in *DS-FP* is not fixed. Theoretically, the separation of two consecutive jobs of τ_i in *DS-FP*, $r_{i,j} - r_{i,j-1}$, satisfies the following condition:

$$\mathcal{V}_i - C_i \geq r_{i,j} - r_{i,j-1} \geq \mathcal{V}_i - \mathit{WCRT}_i \quad (j \geq 1), \quad (9)$$

where WCRT_i is the worst-case response time of jobs of τ_i in *DS-FP*. Note that the maximal separation of $J_{i,j}$ and $J_{i,j-1}$ $(j \geq 1)$, $\max_j\{r_{i,j} - r_{i,j-1}\}$, cannot exceed $\mathcal{V}_i - C_i$, which can be obtained when there are no higher-priority preemptions in the execution of jobs $J_{i,j}$s (e.g., the highest priority transaction τ_1 always has separation $\mathcal{V}_1 - C_1$ for $J_{1,j}$ and $J_{1,j-1}$). Thus, the processor utilization for *DS-FP* should be greater than $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$, which is the CPU workload resulting from the maximal separation $\mathcal{V}_i - C_i$ of each transaction.

If $f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$ where $f_{i,0}^{ml}$ is the first job's response time (i.e., the worst-case response time) of τ_i's job in *ML*, *ML* can be regarded as a special case of *DS-FP* in which sampling (or release) time $r_{i,j+1}^{ml}$ and deadline $d_{i,j+1}^{ml}$ $(j \geq 0)$ can be specified as follows:

$$d_{i,j+1}^{ml} = r_{i,j}^{ml} + \mathcal{V}_i, \quad (10)$$

$$r_{i,j+1}^{ml} = d_{i,j+1}^{ml} - (\Theta_i(r_{i,0}^{ml}, f_{i,0}^{ml}) + C_i). \quad (11)$$

It is clear that $\Theta_i(r_{i,0}^{ml}, f_{i,0}^{ml}) + C_i = f_{i,0}^{ml}$ when $r_{i,0}^{ml} = 0$ $(1 \leq i \leq m)$ in *ML*.

Theorem 3.1: Given a synchronous update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ $(1 \leq i \leq m)$, if $(\forall i)$ $f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$ in *ML*, then

$$\mathit{WCRT}_i \leq f_{i,0}^{ml}$$

where WCRT_i and $f_{i,0}^{ml}$ denote the worst-case response time of τ_i in *DS-FP* and *ML*, respectively.

Proof. This can be proved by contradiction. Suppose that τ_k is the highest priority transaction such that $\mathit{WCRT}_k > f_{k,0}^{ml}$ holds in *DS-FP*. Also it is assumed that the response time of $J_{k,n}$ $(n \geq 0)$, $R_{k,n}$, is the worst for τ_k in *DS-FP*. Note that schedules of τ_1 in *ML* and *DS-FP* are the same as in both cases, τ_1 jobs have the same relative deadline (C_1) and separation/period $(\mathcal{V}_1 - C_1)$. Therefore, $1 < k \leq m$ holds.

As $\mathit{WCRT}_k > f_{k,0}^{ml}$, there must be a transaction τ_l such that (a) τ_l has a priority higher than τ_k $(1 \leq l < k)$; (b) at least two consecutive jobs of τ_l, $J_{l,j-1}$ and $J_{l,j}$, overlap with $J_{k,n}$, and (c) the separation of $J_{l,j-1}$ and $J_{l,j}$ satisfies the following condition:

$$r_{l,j} - r_{l,j-1} < \mathcal{V}_l - f_{l,0}^{ml} \ (j > 0), \quad (12)$$

where $\mathcal{V}_l - f_{l,0}^{ml}$ is the period (i.e., separation) of jobs of τ_l in *ML*.

Claim (a) is true because $k > 1$. It is straightforward that if each higher priority transaction of τ_k only has one job overlapping with $J_{k,n}$, then $R_{k,n} \leq f_{k,0}^{ml}$. This implies that Claim (b) is true. Finally, for $(\forall l < k)$ and $J_{l,j-1}$ and $J_{l,j}$ overlapping with $J_{k,n}$, if

$$r_{l,j} - r_{l,j-1} \geq \mathcal{V}_l - f_{l,0}^{ml} \ (j > 0),$$

then $R_{k,n} > f_{k,0}^{ml}$ cannot be true because the amount of preemptions from higher priority transactions received by $J_{k,n}$ in *DS-FP* is no more than that received by $J_{k,0}$ in *ML*. Thus, Claim (c) is also true.

We know that release time $r_{l,j}$ in *DS-FP* is derived as follows:

$$r_{l,j} = d_{l,j} - R_{l,j} \tag{13}$$

where $R_{l,j}$ is the calculated response time of job $J_{l,j}$, i.e., $\Theta_l(r_{l,j}, d_{l,j}) + C_l$. Following Eq. 12 and 13,

$$\begin{aligned} d_{l,j} - R_{l,j} &= r_{l,j} \ \{\text{By Eq. 13}\} \\ &< r_{l,j-1} + \mathcal{V}_l - f_{l,0}^{ml} \ \{\text{By Eq. 12}\} \\ &= d_{l,j} - f_{l,0}^{ml} \ \{\text{By Eq. 3}\} \end{aligned}$$

Finally,

$$R_{l,j} > f_{l,0}^{ml}. \tag{14}$$

Eq. 14 contradicts the assumption that τ_k is the highest priority transaction such that $WCRT_k > f_{k,0}^{ml}$ holds. Therefore, the theorem is proved. □

The following theorem gives a sufficient condition for the schedulability of *DS-FP*.

Theorem 3.2: Given a synchronous update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ $(1 \leq i \leq m)$, if $(\forall i)$ $f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$ in *ML*, then $\mathcal{T}$ is schedulable with *DS-FP*.

Proof. If $f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$, then the worst-case response times of τ_i $(1 \leq i \leq m)$ in *DS-FP*, $WCRT_i$, satisfy the following condition (by Theorem 3.1):

$$WCRT_i \leq f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}.$$

That is, $WCRT_i$ is no more than $\frac{\mathcal{V}_i}{2}$. Because the following three equations hold in *DS-FP* according to Eq. 5 and 6:

$$r_{i,j} = d_{i,j} - R_{i,j}, \tag{15}$$

$$d_{i,j+1} = r_{i,j} + \mathcal{V}_i. \tag{16}$$

$$d_{i,j+1} = r_{i,j+1} + R_{i,j+1}, \tag{17}$$

Replacing $r_{i,j}$ and $d_{i,j+1}$ in Eq. 16 with Eq. 15 and 17, respectively, it follows that

$$r_{i,j+1} + R_{i,j+1} = d_{i,j} - R_{i,j} + \mathcal{V}_i.$$

That is,

$$r_{i,j+1} - d_{i,j} + R_{i,j+1} + R_{i,j} = \mathcal{V}_i. \tag{18}$$

Because

$$R_{i,j+1} + R_{i,j} \leq 2 \cdot WCRT_i \leq \mathcal{V}_i,$$

it follows from Eq. 18 that $r_{i,j+1} - d_{i,j} \geq 0$ holds. This ensures that it is schedulable to schedule two jobs of τ_i in one validity interval $\mathcal{V}_i$ under *DS-FP*. Thus $\mathcal{T}$ is schedulable with *DS-FP*. □

The following corollary states the correctness of *DS-FP*.

Corollary 3.1: Given a synchronous update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ $(1 \leq i \leq m)$, if $(\forall i)$ $f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$ in *ML*, then *DS-FP* correctly guarantees the *temporal validity* of real-time data.

Proof. As deadline assignment in *DS-FP* follows Eq. 3, the largest distance of two consecutive jobs, $d_{i,j+1} - r_{i,j}$ $(j \geq 0)$, does not exceed $\mathcal{V}_i$. The *validity constraint* can be satisfied if all jobs meet their deadlines, which is guaranteed by Theorem 3.2. □

If $\mathcal{T}$ can be scheduled by *ML*, then by *ML* definition $(\forall i)$ $f_{i,0}^{ml} \leq \frac{\mathcal{V}_i}{2}$. Thus Corollary 3.2, which states a sufficient schedulability condition for *DS-FP*, directly follows from Theorem 3.2.

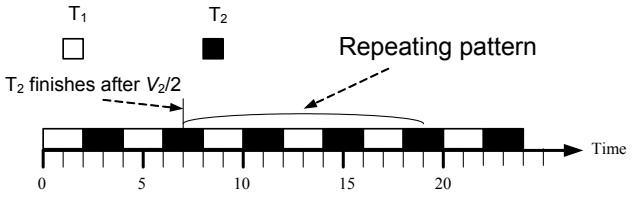
Corollary 3.2: Given a synchronous update transaction set $\mathcal{T}$ with known C_i and $\mathcal{V}_i$ $(1 \leq i \leq m)$, if $\mathcal{T}$ can be scheduled by *ML*, then it can also be scheduled by *DS-FP*.

However, the converse statement of Corollary 3.2 is not true. That is, if $\mathcal{T}$ can be scheduled by *DS-FP*, then it is not necessarily true that $\mathcal{T}$ can also be scheduled by *ML*. This is demonstrated in the following examples.

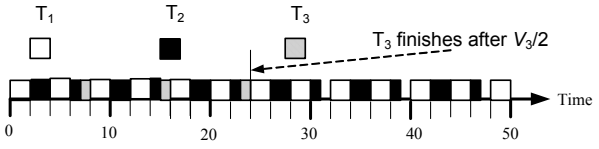
Example 3.3: Consider a set of two transactions $\{\tau_1, \tau_2\}$ with computation times 2, 3, and validity intervals 6, 12, respectively. Figure 5(a) shows a *DS-FP* schedule for this transaction set. This schedule is valid because the pattern between time 7 and 19 repeats itself forever. On the other hand, this transaction set is not schedulable by *ML* because the first job of τ_2, $J_{2,0}$, completes at time 7, which is greater than $\frac{\mathcal{V}_2}{2}$ (i.e., 6).

In this example, *DS-FP* works better because it allows $J_{2,0}$ to be completed later than $\frac{\mathcal{V}_2}{2}$. □

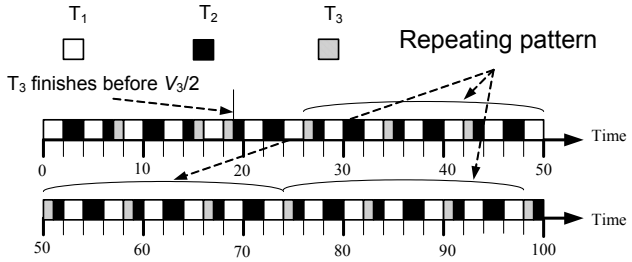
Example 3.4: Consider a set of three transactions $\{\tau_1, \tau_2, \tau_3\}$ with computation times 2, 3, 3, and validity intervals 6, 15, 47, respectively. Figure 5(b) depicts a schedule of the transactions under *ML*. The first job of τ_3, $J_{3,0}$, completes at time 24, which is greater than $\frac{\mathcal{V}_3}{2}$ (i.e., 23.5). Thus the set of transactions is not schedulable by *ML*. Figure 5(c) depicts a



(a) A *DS-FP* schedule for 2 transactions that ML fails



(b) ML fails 3 transactions



(c) A *DS-FP* schedule for 3 transactions that ML fails

Figure 5. Transaction sets schedulable by *DS-FP* but not *ML*

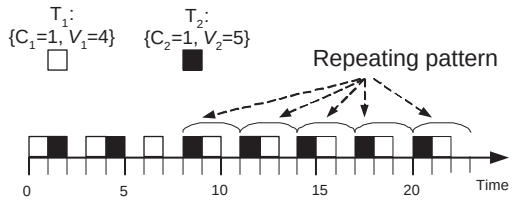
schedule of the transactions under *DS-FP*. The same transaction set is schedulable by *DS-FP* because the schedule pattern between time 26 and 50 repeats itself forever.

Different from Example 3.3, it is observed in this *DS-FP* schedule that the first jobs of all transactions finish before half of their respective validity intervals. □

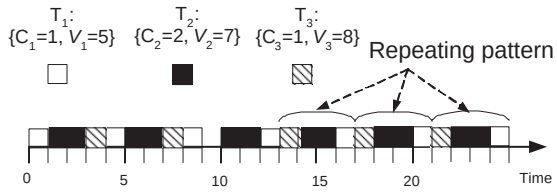
Note that in these two examples, *DS-FP* fully utilizes the processor. We could easily derive examples in which the processor idles once in a while. For example, in Figure 5(a) we could change C_2 to 2.5 and in Figure 5(b) we could change C_3 to 2.75. After both changes the transaction sets cannot be scheduled by *ML*. However, they can be scheduled by *DS-FP*. Moreover,we can also scale up the numbers to make them all integers again.

In summary, if a set of synchronous update transactions can be scheduled by *ML* to satisfy the *validity constraint*, then it can also be scheduled by *DS-FP*. However, the converse statement is not true, which implies that *DS-FP outperforms ML in terms of schedulability*. Thus, the following corollary directly follows from both Corollary 3.2 and Example 3.4.

Corollary 3.3: *DS-FP outperforms ML in terms of schedulability* for satisfying the *validity constraint*.



(a) A *DS-FP* schedule for 2 transactions



(b) A *DS-FP* schedule for 3 transactions

Figure 6. *DS-FP* schedules with *fixed patterns*

Discussion of jitters: Our results can be easily extended to the case that jitter between sampling time and release time of a job is non-zero if the maximum jitter of a transaction is known. In the presence of non-zero jitters, we can transform a transaction τ_i' (with validity length $\mathcal{V}_i'$ and maximum jitter δ_i) to a transaction τ_i (with validity length $\mathcal{V}_i = \mathcal{V}_i' - \delta_i$ and zero jitter). Such a transformation guarantees that if τ_i can meet its validity constraint, then τ_i' can also meet its validity constraint.

3.4 Theoretical Estimation of Processor Utilization for *DS-FP*

This subsection presents means of estimating average CPU utilization. Note that *DS-FP* does not usually schedule transactions periodically. Thus, it is hard to derive its exact CPU utilization unless there is a fixed pattern that repeats itself in a *DS-FP* schedule. In what follows, we shall investigate two cases in order: (1) a *DS-FP* schedule that has a detected pattern repeating itself from certain point in time; (2) a *DS-FP* schedule that has no detected pattern.

3.4.1 *DS-FP* with a Detected Pattern

We introduce *fixed pattern* in a *DS-FP* schedule with a simple example, which is shown below.

Example 3.5: Consider a *DS-FP* schedule for two transactions τ_1 and τ_2 in Figure 6(a). Note that transaction parameters (C_i's and $\mathcal{V}_i$'s) are given in the figure. We observe that there is a *fixed pattern* repeating itself in the schedule every 3 time units, starting from time 8. If time goes to infinity, we can estimate that the average CPU utilization of the *DS-FP* schedule is about 66.7%. Similarly, given three transactions in Figure 6(b), we observe a *fixed pattern* repeating itself in the schedule every 4 time units, starting from time 13. Again, we can easily estimate that its CPU utilization is close to 100%. □

Needless to say, the average CPU utilization for a *DS-FP* schedule can be approximated based on a *fixed pattern* if such a pattern exists in the schedule. However, it is not true that we can always easily detect a fixed pattern in every *DS-FP* schedule. It becomes harder to detect a *fixed pattern* in a *DS-FP* schedule if the size of the transaction set is larger. This is because the complexity of pattern detection grows exponentially with the size of the transaction set. Indeed, it remains open whether there is always a fixed pattern in every *DS-FP* schedule. As many *DS-FP* schedules may not be detected to have such *fixed pattern*s, it becomes more important to estimate the average CPU utilization for such *DS-FP* schedules.

3.4.2 *DS-FP* without a Detected Pattern

We now present an approximation of average processor utilization of *DS-FP* from statistical perspective *in the absence of detected patterns in* DS-FP *schedules*. Note that *our approximation only works provided that $\mathcal{T}$ can be scheduled by* ML. This implies that the approximation is applicable to transaction sets that all deadlines are not greater than their corresponding periods in *ML*. Our approximation is quite close to the average CPU utilization obtained in our experiments. The CPU utilization approximation depends on the approximate values of the average deadline $\overline{D}$ and period $\overline{P}$ of transactions, which is described as follows.

Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^m$, let $\overline{U}_{DS}$ denote the average processor utilization in *DS-FP*, and $\overline{P}_j$ the average period for τ_j. The average relative deadline of τ_i, namely $\overline{D}_i$, is approximated as follows:

$$\overline{D}_i = C_i + \sum_{j=1}^{i-1}[(\frac{\overline{D}_i}{\overline{P}_j}) \times C_j] \quad (1 \leq i \leq m). \tag{19}$$

Let $P_{i,j}$ and $D_{i,j+1}$ $(1 \leq i \leq m \ \wedge \ j \geq 0)$ denote $r_{i,j+1} - r_{i,j}$ and $d_{i,j+1} - r_{i,j+1}$ in Eq. 4, respectively. It follows that

$$P_{i,j} + D_{i,j+1} = \mathcal{V}_i. \tag{20}$$

Thus the following equation holds given an arbitrarily large n $(n \to \infty)$, where n is the number of jobs in averaging:

$$\overline{P}_i + \overline{D}_i = \mathcal{V}_i. \tag{21}$$

Following Eq. 19 and 21, $\overline{D}_i$ and $\overline{P}_i$ $(1 \leq i \leq m)$ can be calculated (from the highest priority transaction τ_1 to the lowest priority transaction τ_m) as following, respectively:

$$\overline{D}_i = \frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{\overline{P}_j}} \quad (1 \leq i \leq m) \tag{22}$$

$$\overline{P}_i = \mathcal{V}_i - \overline{D}_i \quad (1 \leq i \leq m) \tag{23}$$

Finally, $\overline{U}_{DS}$, the average utilization of the transaction set $\mathcal{T}$ under *DS-FP* can be approximated as:

$$\overline{U}_{DS} = \sum_{i=1}^{m} \frac{C_i}{\overline{P}_i} = \sum_{i=1}^{m} (\frac{C_i}{\mathcal{V}_i - \frac{C_i}{1 - \sum_{j=1}^{i-1} \frac{C_j}{\overline{P}_j}}}) \tag{24}$$

The following example illustrates how the average utilization is estimated.

Example 3.6: Given the transaction set in Table 2, we calculate the average relative deadline and period of τ_i $(i = 1, 2, 3)$ as follows:

$$\overline{D}_1 = C_1 = 1, \ \overline{P}_1 = \mathcal{V}_1 - \overline{D}_1 = 4,$$
$$\overline{D}_2 = \frac{C_2}{1 - \frac{C_1}{\overline{P}_1}} = 2.7, \ \overline{P}_2 = \mathcal{V}_2 - \overline{D}_2 = 7.3,$$
$$\overline{D}_3 = \frac{C_3}{1 - (\frac{C_1}{\overline{P}_1} + \frac{C_2}{\overline{P}_2})} = 4.2, \ \overline{P}_3 = \mathcal{V}_3 - \overline{D}_3 = 15.8.$$

The average processor utilization is $\overline{U}_{DS} = \sum_{i=1}^{m} \frac{C_i}{\overline{P}_i} = 0.65$. Given the transaction set in Table 2, it can be verified that the processor utilization for the first 200 time units is 63%, which is very close to our theoretical estimation and lower than the processor utilization from *ML* (68%). □

Discussion of fixed patterns: A *fixed pattern* in a *DS-FP* schedule may be exponentially long (with respect to the number of transactions). Thus, it can be very expensive to detect. Assume that the minimal number of jobs per transaction in this pattern is n. If n is large, then Eq. 24 can be used to estimate the average CPU utilization of the fixed pattern, which in turn is the utilization estimation of the schedule.

In summary, the average CPU utilization of a *DS-FP* schedule can be approximated based on a fixed pattern if such a pattern exists in the schedule. Otherwise, the CPU utilization can be estimated by Eq. 24 if the transaction set is schedulable according to Corollary 3.2.

3.5 The Non-Optimality of *DS-FP*

We have proven in Section 3.4 that *DS-FP* is close to optimal in terms of minimizing CPU workload from the statistical perspective. Intuitively, *DS-FP* should be very close to an optimal algorithm because it always defers the execution of a job as late as possible, and hence reducing the workload as much as possible. We have also proven that *DS-FP* can schedule any transaction set that is schedulable by *ML* in Section 3.3. Now it is interesting to know if *DS-FP* is an optimal algorithm in terms of schedulability. That is, *given any transaction set, if it is schedulable by a fixed priority scheduler, can it be scheduled by* DS-FP? Unfortunately, the answer to the aforementioned question is negative, which can be demonstrated with the following example.

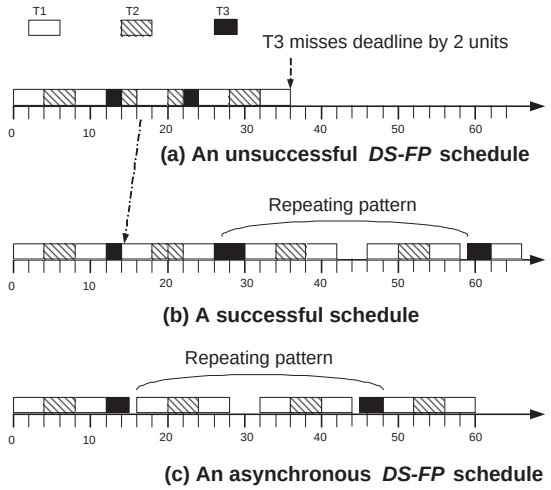


Figure 7. *DS-FP* is not optimal

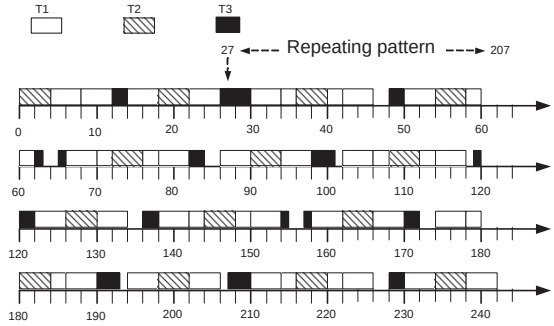


Figure 8. *SVF* is not optimal for *DS-FP*

Example 3.7: Consider a set of three transactions $\{\tau_1, \tau_2, \tau_3\}$ with computation times 4, 4, 3, and validity intervals 12, 22, 36, respectively. This set is not schedulable by *DS-FP* as it fails at time 36, shown in Figure 7(a). In this case, the second job of τ_3 cannot be completed by the end of its first validity interval. However, if $J_{1,2}$ is scheduled 2 time units earlier, this transaction set can be successfully scheduled because there is a *fixed pattern* repeating itself every 32 time units starting from time 27, as depicted in Figure 7(b). Note that such a schedule is also a fixed priority schedule because no lower priority jobs may interrupt a higher priority job once the higher one is released. By doing so, the release time of $J_{2,1}$ is postponed to time 18 as shown in Figure 7(b) (from time 14 in Figure 7(a)). Hence the deadline of $J_{2,2}$ is also postponed. □

DS-FP requires that every transaction τ_i should finish its first two jobs in $[0, \mathcal{V}_i)$. If the requirement is relaxed so that the first two jobs are allowed to finish in $[r_{i,0}, r_{i,0} + \mathcal{V}_i)$ where $r_{i,0}$ denotes the time at which $J_{i,0}$ actually starts, then *DS-FP* can schedule the set in Example 3.7. In this case, the first jobs of transactions start *asynchronously*. An asynchronous schedule for the same transaction set in Example 3.7 is depicted in Figure 7(c), in which there is a *fixed pattern* between time 16 and 48 repeating itself forever. In general, whether the asynchronous *DS-FP* algorithm is optimal in terms of schedulability remains an open question.

Another interesting observation is that the transaction set in Example 3.7 is schedulable by *DS-FP* if a priority order different from *Shortest Validity First* (*SVF*) is used. For example, if we swap the priorities of τ_1 and τ_2, *DS-FP* can schedule the set, as depicted in Figure 8. In this case there is a *fixed pattern* between time 27 and 207 repeating itself forever.

In summary, *DS-FP* is not optimal for a set of synchronous update transactions in terms of schedulability. But it remains open if it is optimal for asynchronous transactions, or transaction priorities assigned differently from SVF.

Parameters	Meaning	Value
N_{CPU}	No. of CPU	1
N_T	No. of real-time data objects	$[10, 300]$
$\mathcal{V}_i$ (ms)	Validity interval of data X_i	$[4000, 8000]$
C_i (ms)	CPU time for updating X_i	$[5, 15]$
Length	No. of data to update	1

Table 4. Experimental parameters and settings

4 PERFORMANCE EVALUATION

This section presents the important results from our simulation studies. Section 4.1 describes our simulation model and parameters. Section 4.2 compares *DS-FP* with the More-Less (*ML*) algorithm. *ML* is known to outperform Half-Half [25], which is not compared here. The experiments demonstrate that our proposed approaches reduce CPU utilization while guaranteeing data validity constraints.

4.1 Simulation Model and Parameters

Our experiments compare the update transaction workloads produced by *DS-FP* and *ML*. It is demonstrated that *DS-FP* produces a lower CPU workload than *ML*. Also, the experiments demonstrate that the increase of average sampling period from *DS-FP* is the main reason for its lower workload. The primary performance metric used in the experiments is CPU workload.

A summary of the parameters and default settings used in experiments is presented in Table 4. The baseline values for the parameters follow those used in [25], which are originally from air traffic control applications. We consider a single CPU, main memory based RTDBS. The number of real-time data objects varies from 10 to 300 and the valid-

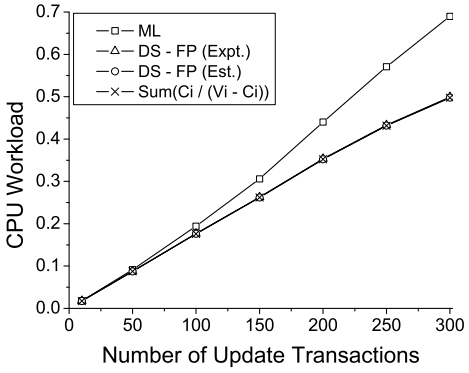


Figure 9. CPU workloads comparison

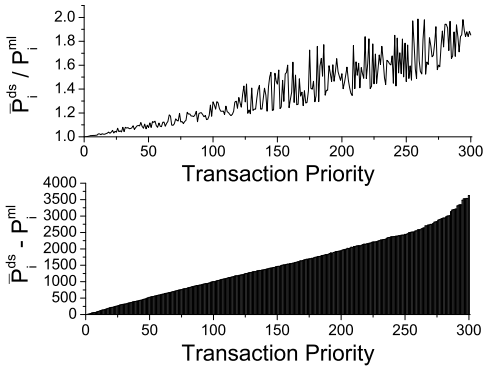


Figure 10. Average sampling period comparison

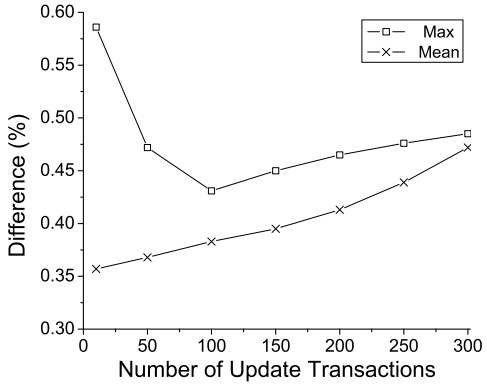


Figure 11. CPU workloads estimation error

ity interval of each real-time data object is uniformly distributed between 4000 ms and 8000 ms. Each transaction updates one real-time data object, and the CPU time for each transaction is uniformly distributed between 5 ms and 15 ms. In the experiments, 95 percent confidence intervals have been obtained whose widths are less than ± 5 percent of the point estimate for the performance metrics.

4.2 Experimental Results

In our experiments, the CPU workloads of update transactions produced by *ML* and *DS-FP* are quantitatively compared. Update transactions are generated randomly according to the parameter settings in Table 4.

The resulting CPU workloads generated from *ML* and *DS-FP* are depicted in Figure 9. From the results, we observe that the CPU workload produced by *DS-FP* is consistently lower than that of *ML*. In fact, the difference widens as the number of update transactions increases. The difference reaches 18% when the number of transactions is 300. It is also observed that the CPU utilization of *DS-FP* measured in our experiments (*DS-FP*(Expt.)) nearly matches the CPU workload estimation $\overline{U}_{DS}$ (Eq. 24), shown as *DS-FP*(Est.) in the figure. Moreover, the *DS-FP* CPU workload is only slightly higher than $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$, which is the CPU workload resulting from the maximal separation $\mathcal{V}_i - C_i$ $(1 \leq i \leq m)$ of each transaction (see Section 3.3). In fact, the difference is insignificant in Figure 9. The improvement of the CPU workload in *DS-FP* is due to the fact that *DS-FP* adaptively samples real-time data objects at a lower rate. This is verified by the average sampling periods of update transactions obtained from experiments.

Figure 10 shows the average sampling period for each transaction in *DS-FP* when the number of update transactions is 300. Given a set of update transactions, the period of transaction τ_i in *ML* (P_i^{ml}) is a constant and it can be calculated off-line [25], while the separation of sampling times of two consecutive jobs from the same transaction in *DS-FP* is dynamic and it is obtained on-line in the experiments. The mean value of the separations, i.e., the average sampling period, $\overline{P}_i^{ds}$, for transaction τ_i is calculated as follows, where n is the number of jobs generated by τ_i in the experiments.

$$\overline{P}_i^{ds} = \frac{1}{n-1} \sum_{j=1}^{n-1} (r_{i,j} - r_{i,j-1}) \tag{25}$$

In Figure 10, it is observed that $\overline{P}_i^{ds}$ is consistently larger than P_i^{ml} while the difference ($\overline{P}_i^{ds} - P_i^{ml}$) increases with the decreasing of the transaction's priority. *DS-FP* reduces the average sampling rate more for lower-priority transactions, thus reducing the workload of CPU. Figure 10 also shows that the trend of ($\frac{\overline{P}_i^{ds}}{P_i^{ml}}$) increases similarly to that of ($\overline{P}_i^{ds} - P_i^{ml}$), although it fluctuates.

Figure 11 depicts how much the CPU workload estimation (*DS-FP*(Est.)) differs from the actual CPU utilization obtained from the experiments (*DS-FP*(Expt.)) in finer granularity. The x-axis depicts the size of update transactions and the y-axis depicts the relative difference between *DS-FP*(Est.) and *DS-FP*(Expt.), which is defined as

$$\frac{\mid \textit{DS-FP}(Expt.) - \textit{DS-FP}(Est.) \mid}{\textit{DS-FP}(Expt.)} \times 100\%.$$

Both maximum and mean relative differences are depicted in the figure. In our experiments, it is observed that *DS-*

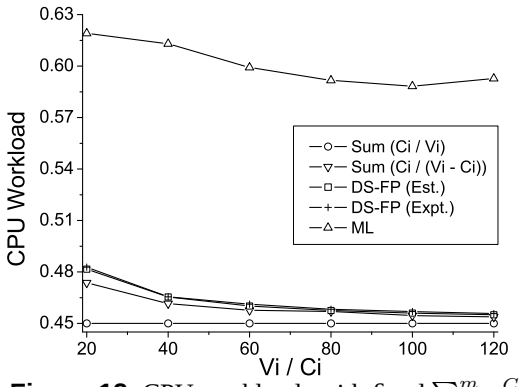


Figure 12. CPU workloads with fixed $\sum_{i=1}^{m} \frac{C_i}{V_i}$

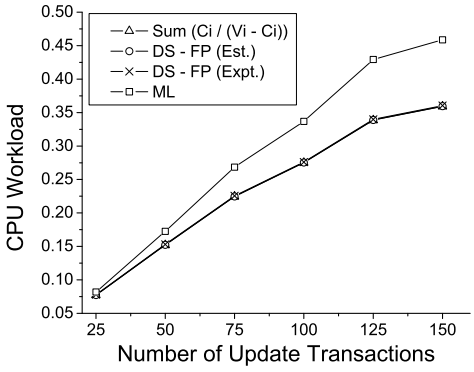


Figure 13. CPU workloads comparison

FP(Expt.) is consistently higher than *DS-FP*(Est.). As observed from the figure, our CPU workload estimation nearly matches the measured CPU utilization in our experiments as the maximum relative difference never exceeds 0.6%.

We also conducted a set of experiments by varying $\frac{\mathcal{V}_i}{C_i}$ and fixing $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i}$ of the update transaction set at 45%. The results are depicted in Figure 12, which compares *ML*, *DS-FP*, $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i}$ and $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$. Similar to Figure 9, the actual utilization for *DS-FP* is very close to the utilization estimation $\overline{U}_{DS}$ (shown as *DS-FP*(Est.)). Note that $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$ is the CPU workload resulting from the possible maximum separation $\mathcal{V}_i - C_i$ satisfying the *validity constraint* for each transaction τ_i. It is a CPU lower bound ignoring transaction interference. It is observed in Figure 12 that a CPU workload of *DS-FP* is very close to that of $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$. The larger $\frac{\mathcal{V}_i}{C_i}$ is, the closer *DS-FP* and $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i - C_i}$ are. This is because the probability of transaction interference decreases for *DS-FP* when $\frac{\mathcal{V}_i}{C_i}$ increases.

We have conducted more experiments with different experimental settings to verify whether *DS-FP* is sensitive to parameters. In Figure 13, we vary the validity length ($\mathcal{V}_i$) from 1000 to 8000 to have more spread on validity length. In Figure 14, while we vary the validity length from 1000 to 8000, we also vary the computation time (C_i) from 5 to 50 so that a single transaction may have higher CPU utilization. It is demonstrated in both figures that our observations from Figure 9 are well maintained under various parameter settings. Indeed, these results demonstrate that the improvement of the CPU workload in *DS-FP* is due to the fact that *DS-FP* adaptively samples real-time data objects at a lower rate. This again is verified by the average sampling periods of update transactions obtained from experiments.

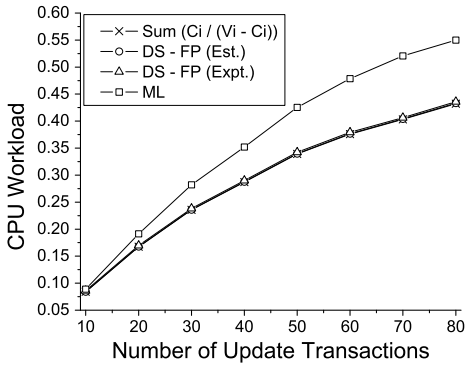


Figure 14. CPU workloads comparison

In summary, when a set of update transactions is scheduled by *DS-FP* to maintain the temporal validity of real-time data objects, it produces a schedule with a much lower CPU workload than *ML* does. Thus more CPU capacity is available for improving the performance of other workloads (e.g., triggered transactions [27]) in the system.

5 RELATED WORK

There has been a lot of work on RTDBSs in which validity intervals are associated with real-time data [21, 22, 1, 12, 13, 14, 5, 26, 11, 30, 8, 7, 25, 10]. In [6], a safety-critical automotive application, adaptive cruise control, is studied. It deals with critical data and involves deadline bound computations on data gathered from the automobiles' environment. These applications have stringent requirements on the freshness of data objects and completion times of the tasks. In [8], a vehicular application with embedded engine control systems is presented, and an on-demand scheduling algorithm is proposed for enforcing base and derived data freshness. In [7], an algorithm (ODTB) is proposed for updating data items that can skip unnecessary updates, allowing for better utilization of the CPU in the vehicular application. Such systems introduce the need to maintain data temporal consistency in addition to logical consistency.

In the model introduced in [22], a real-time system consists of periodic tasks which are either read-only, write-only or update (read-write) transactions. Data objects are temporally inconsistent when their ages or dispersions are greater than the absolute or relative thresholds allowed by the application. Two-phase locking and optimistic concurrency control algorithms, as well as rate-monotonic and earliest deadline first scheduling algorithms are studied in [22]. In

[12, 13], real-time data semantics are investigated and a class of real-time data access protocols called SSP (Similarity Stack Protocols) is proposed. The correctness of SSP is based on the concept of similarity, which allows different but sufficiently timely data to be used in a computation without adversely affecting the outcome.

In [14], similarity-based principles are coupled with the *Half-Half* approach to adjust the real-time transaction load by skipping the execution of task instances. The concept of *data-deadline* is proposed in [26]. It also proposes data-deadline based scheduling, forced-wait and similarity-based scheduling techniques to maintain the temporal validity of real-time data and to meet transaction deadlines in RTDBSs. In [10], Jha et al. study whether, given an update transaction schedule, a periodic query would read mutually consistent data. They propose design approaches to decide the period and relative deadline of a query so that it satisfies mutual consistency. They then suggest ways of reducing the complexity of the solution approach using harmonic periods in general.

Our work is related to the *ML* scheme in [2, 25, 30]. *ML* guarantees a bound on the sampling time of a periodic transaction job and the finishing time of its next job. But, as we showed, the deadline and period of a periodic transaction are derived from the worst-case response time of the transaction. This is different from the aperiodic task model based *DS-FP* algorithm in which the deadline of a transaction job is derived adaptively, and the separation of two consecutive jobs is not a constant. *DS-FP* reduces the CPU workload resulting from update transactions further by adaptively adjusting the separation of two consecutive jobs while satisfying the *validity constraint*. *DS-FP* is also different from the *distance constrained scheduling*, which guarantees a bound of the finishing times of two consecutive instances of a task [9]. The *EDL* algorithm proposed in [4] processes tasks as late as possible based on the Earliest Deadline scheduling algorithm [17]. However, *EDL* assumes that all deadlines of tasks are given whereas *DS-FP* derives deadlines dynamically. Finally, our *DS-FP* algorithm is applicable to the scheduling of *age constraint* tasks in real-time systems [24].

6 CONCLUSIONS AND FUTURE WORK

This article proposes a novel algorithm, namely deferrable scheduling for fixed priority transactions (*DS-FP*). Distinct from past studies of maintaining the freshness (or temporal validity) of real-time data in which the periodic task model is adopted, *DS-FP* adopts the *aperiodic* task model. The deadlines of jobs and the separation of two consecutive jobs of an update transaction are adjusted judiciously so that the farthest distance of the sampling time of a job is achieved and the completion time of its next job is bounded by the validity length of the updated real-time data. This article presents a sufficient condition for its schedulability. It also proposes a theoretical estimation of the processor utilization of *DS-FP*, which is verified in our experimental studies. It is also demonstrated in our experiments that *DS-FP* greatly reduces update workload compared to *ML* while guaranteeing the validity constraint.

However, there are still many open questions to be answered for *DS-FP*. For example, it is not clear what a sufficient and necessary condition is for schedulability of *DS-FP*, if time 0 is a critical instant for a synchronous transaction set scheduled by *DS-FP*, and if there is a least upper bound of CPU utilization for *DS-FP*. Moreover, the concept of *deferrable scheduling* is only used to schedule update transactions with fixed priority in this article. It is possible for the same concept to be used in the scheduling of update transactions with dynamic priority, e.g., in the Earliest Deadline scheduling [17, 4] of update transactions.

ACKNOWLEDGMENTS

Preliminary versions of this paper appeared in [27, 28]. The authors would like to thank Professors Krithi Ramamritham and Aloysius K. Mok for fruitful discussions on this work.

References

[1] N. C. Audsley, A. Burns, M. F. Richardson, A. J. Wellings, "Data Consistency in Hard Real-Time Systems," Informatica 19(2), 1995.

[2] A. Burns and R. Davis, "Choosing task periods to minimise system utilisation in time triggered systems," in *Information Processing Letters*, 58 (1996), pp. 223-229.

[3] D. Chen and A. K. Mok, "Scheduling Similarity-Constrained Real-Time Tasks," pp. 215-221, *ESA/VLSI*, 2004.

[4] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, pp. 1261-1269, October 1989.

[5] R. Gerber, S. Hong and M. Saksena, "Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes," *IEEE Real-Time Systems Symposium*, December 1994.

[6] G. Goud, N. Sharma, K. Ramamritham, and S. Malewar. "Efficient Real-Time Support for Automotive Applications: A Case Study," In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2006.

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

[8] T. Gustafsson and J. Hansson, "Dynamic on-demand updating of data in real-time database systems," *ACM SAC*, 2004.

[9] C. C. Han, K. J. Lin and J. W.-S. Liu, "Scheduling Jobs with Temporal Distance Constraints," *SIAM Journal of Computing*, Vol. 24, No. 5, pp. 1104 - 1121, October 1995.

[10] A. K. Jha, M. Xiong, K. Ramamritham, "Mutual Consistency in Real-Time Databases," *IEEE Real-Time Systems Symposium*, December 2006.

[11] K. D. Kang, S. Son, J. A. Stankovic, and T. Abdelzaher, "A QoS-Sensitive Approach for Timeliness and Freshness Guarantees in Real-Time Databases," *EuroMicro Real-Time Systems Conference*, June 2002.

[12] T. Kuo and A. K. Mok, "Real-Time Data Semantics and Similarity-Based Concurrency Control," *IEEE Real-Time Systems Symposium*, December 1992.

[13] T. Kuo and A. K. Mok, "SSP: a Semantics-Based Protocol for Real-Time Data Access," *IEEE Real-Time Systems Symposium*, December 1993.

[14] S. Ho, T. Kuo, and A. K. Mok, "Similarity-Based Load Adjustment for Real-Time Data-Intensive Applications," *IEEE Real-Time Systems Symposium*, 1997.

[15] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," *IEEE Real-Time Systems Symposium*, 1990.

[16] J. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks," *Performance Evaluation*, 2(1982), 237-250.

[17] C. L. Liu, and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, 20(1), 1973.

[18] D. Locke, "Real-Time Databases: Real-World Requirements," in *Real-Time Database Systems: Issues and Applications*, edited by Azer Bestavros, Kwei-Jay Lin and Sang H. Son, Kluwer Academic Publishers, pp. 83-91, 1997.

[19] K. Ramamritham, "Real-Time Databases," *Distributed and Parallel Databases* 1(1993), pp. 199-226, 1993.

[20] K. Ramamritham, "Where Do Time Constraints Come From and Where Do They Go ?" International Journal of Database Management, Vol. 7, No. 2, Spring 1996, pp. 4-10.

[21] X. Song and J. W. S. Liu, "How Well Can Data Temporal Consistency be Maintained?" *Proceedings of IEEE Symposium on Computer-Aided Control Systems Design*, 1992.

[22] X. Song and J. W. S. Liu, "Maintaining Temporal Consistency: Pessimistic vs. Optimistic Concurrency Control," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 5, pp. 786-796, October 1995.

[23] J. A. Stankovic, S. Son, and J. Hansson, "Misconceptions About Real-Time Databases," *IEEE Computer*, Vol. 32, No. 6, pp. 29-36, June 1999.

[24] Steve Vestal, "Real-Time Sampled Signal Flows through Asynchronous Distributed Systems," *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 170-179, 2005.

[25] M. Xiong and K. Ramamritham, "Deriving Deadlines and Periods for Real-Time Update Transactions," *IEEE Real-Time Systems Symposium*, 1999.

[26] M. Xiong, K. Ramamritham, J. A. Stankovic, D. Towsley, and R. M. Sivasankaran, "Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics," *IEEE Transactions on Knowledge and Data Engineering*, 14(5), 1155-1166, 2002.

[27] M. Xiong, S. Han, and K.Y. Lam, "A Deferrable Scheduling Algorithm for Real-Time Transactions Maintaining Data Freshness," *IEEE Real-Time Systems Symposium*, December 2005.

[28] M. Xiong, S. Han, and D. Chen, "Deferrable Scheduling for Temporal Consistency: Schedulability Analysis and Overhead Reduction," *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2006.

[29] M. Xiong, S. Han, D. Chen, and K.Y. Lam, "Deferrable Scheduling for Maintaining Real-Time Data Freshness: Algorithms, Analysis, and Results," *Technical Report, The University of Texas at Austin, Departments of Computer Sciences*, TR-07-44, September 2007.

[30] M. Xiong, B. Liang, K.Y. Lam, and Y. Guo, "Quality of Service Guarantee for Temporal Consistency of Real-time Transactions," *IEEE Transactions on Knowledge and Data Engineering*, 18(8), pp. 1097-1110, 2006.