A Deferrable Scheduling Algorithm for Real-Time Transactions Maintaining Data Freshness

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

Song Han
City University of Hong Kong
han_song@cs.cityu.edu.hk

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

Abstract

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 [22] in which update transactions are guaranteed by the Deadline Monotonic scheduling algorithm [13] to complete by their deadlines. In this paper, we propose a novel algorithm, namely deferrable scheduling, for minimizing imposed workload while maintaining temporal validity of real-time data. In contrast to previous work, update transactions scheduled by the deferrable scheduling algorithm follow a sporadic task model. 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 analysis of its processor utilization, which is verified in our experiments. Our experimental results also demonstrate that the deferrable scheduling algorithm is a very effective approach, and it significantly outperforms the More-Less scheme in terms of reducing processor workload.

1 Introduction

Real-time and embedded systems are used 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 an air traffic control system, and temperature as well as air pressure in an engine control environment. Such real-time data are typically stored in a real-time database system (RTDBS). They are used to model the current status of entities in a system environment. However, real-time data are different from traditional data stored in databases in that they have time semantics indicating that sampled values are valid only for a certain time interval [17, 15, 20]. The concept of *temporal validity* is first introduced in [17] to define the correctness of real-time data. 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 its old value expires, i.e., the old one becomes temporally invalid. Otherwise, the RTDBS cannot detect and respond to environmental changes timely. In recent years, there has been tremendous amount of work devoted to this area [4, 9, 11, 12, 17, 18, 19, 21, 8, 22, 6].

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 objects periodically [17, 11, 22]. A sensor update transaction has an infinite number of periodic jobs, which have fixed length period and relative deadline. The update problem for periodic update transactions consists of two parts [22]: *(1) the determination of the sampling periods and deadlines of update transactions; and (2) the scheduling of update transactions*. Two methods have been proposed in prior work for minimizing the update workload while maintaining the data freshness. As explained in [17, 11], a simple method to maintain temporal validity of real-time data objects 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 the validity interval length of the object. To further reduce the update workload, the *More-Less (ML)* scheme is proposed [22].

This paper presents our *Deferrable Scheduling* algorithm for *Fixed Priority* transactions (*DS-FP*) with the objective to minimize the update workload. 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 has a fixed period and relative deadline for each transaction, *DS-FP* adopts a *sporadic* 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 to be updated. The deferral of a job's sampling time results in a relative deadline that is less than its worst-case response time, thus increases the separation of two consecutive jobs. This helps reducing processor workload produced by update transactions. The average processor utilization under *DS-FP* is theoretically analyzed. 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*.

The rest of the paper is organized as follows: Section 2 reviews the existing approaches for freshness maintenance of real-time data. In Section 3, the Deferrable Scheduling algorithm for Fixed Priority transactions (*DS-FP*) is proposed and discussed in detail. Section 4 presents the performance studies and Section 5 briefly describes the related work. Finally, we conclude our study in Section 6.

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 value 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 [17, 18, 15]. 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 absolutely consistent) if its j^{th} sampling time ($r_{i,j}$) plus the validity interval ($\mathcal{V}_i$) of the data object is not less than t, i.e., $r_{i,j} + \mathcal{V}_i \geq t$ [17].

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 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 paper, $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ 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. All real-time data objects

Symbol	Definition
X_i	Real-time data object i
τ_i	Update transaction updating X_i
$J_{i,j}$	The jth job of τ_i
$R_{i,j}$	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 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*

Table 1. Symbols and definitions.

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 the 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. For convenience, let $d_{i,j}, f_{i,j}$ and $r_{i,j}$ denote the absolute deadline, completion time, and sampling (release) time of the j^{th} job $J_{i,j}$ of τ_i, respectively. We also assume that jitter between sampling time and release time of a job is zero (readers are referred to [22] 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 *periodic* task model, is to determine period P_i and relative deadline D_i so that all the update transactions are schedulable and 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 once every period. However, there is no guarantee on when a job of a periodic transaction is actually executed within a period. Throughout this paper we assume the scheduling algorithms are preemptive and ignore all preemption overhead. For convenience, we use terms transaction and task interchangeably in this paper.

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 [17, 11]. 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. Un-

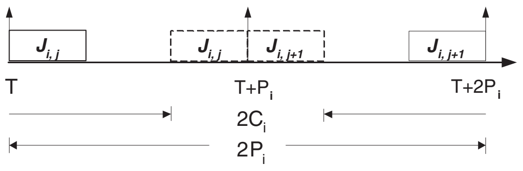
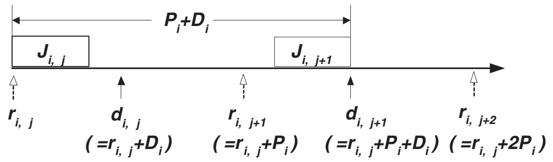


Figure 1. Extreme execution cases of jobs $J_{i,j}$ and $J_{i,j+1}$



P$_i$+D$_i$
$J_{i,j}$
$J_{i,j+1}$
$r_{i,j}$
$d_{i,j}$ $(=r_{i,j}+D_i)$
$r_{i,j+1}$ $(=r_{i,j}+P_i)$
$d_{i,j+1}$ $(=r_{i,j}+P_i+D_i)$
$r_{i,j+2}$ $(=r_{i,j}+2P_i)$

Figure 2. Illustration of *More-Less* scheme

fortunately, this approach incurs unnecessarily high CPU workload of the update transaction 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.

Lemma 2.1: For a set of periodic transactions $\mathcal{T} = \{\tau_i\}_{i=1}^m$ $(D_i \le P_i)$ 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 $I_1 = I_2 = ... = I_m = 0$. [13] □

For $I_i = 0$ $(1 \le i \le m)$, 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, *ML* is proposed to guarantee temporal validity [22], in which *Deadline Monotonic* (*DM*) [13] is used to schedule periodic update transactions. There are three constraints to follow for τ_i $(1 \le i \le m)$:

- *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., $P_i + D_i \le \mathcal{V}_i$, 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., $C_i \le D_i \le P_i$.
- *Feasibility constraint*: for a given set of update transactions, *Deadline Monotonic* scheduling algorithm [13] is used to schedule the transactions. Consequently, $\sum_{j=1}^{i}(\lceil \frac{D_i}{P_j} \rceil \cdot C_j) \le D_i$ $(1 \le i \le m)$ if τ_i has higher priority than τ_j 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:

$$D_i = f_{i,0}^{ml} - r_{i,0}^{ml}, \tag{1}$$

$$P_i = \mathcal{V}_i - D_i, \tag{2}$$

where $f_{i,0}^{ml}$ and $r_{i,0}^{ml}$ are finishing and sampling times of the first job of τ_i, 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 paper, superscript *ml* is used to distinguish the finishing and sampling times in *ML* from those in *DS-FP*.

3 Deferrable Scheduling

A scheduler is *work-conserving* if it never idles the processor while there is a job awaiting execution. All schedulers discussed in this paper are work-conserving. Next, we present our *Deferrable Scheduling algorithm for Fixed Priority* transactions (*DS-FP*), and compare it with *ML*.

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 periodic task model that has a fixed period and deadline for each task, and the deadline is equivalent to the worst-case response time. It should be noted that the *validity constraint* can always be satisfied as long as $P_i + D_i \le \mathcal{V}_i$. But 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 $J_{i,j+1}$ $(j \ge 0)$,

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

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

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

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. 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

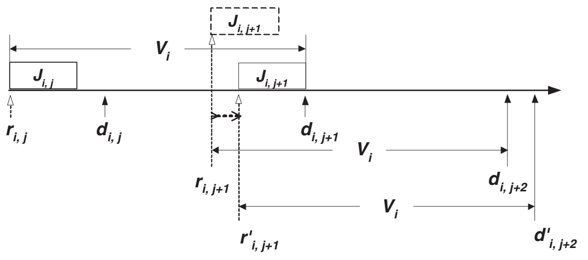


Figure 3. Illustration of *DS-FP* scheduling ($r_{i,j+1}$ is shifted to $r^{'}_{i,j+1}$)

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 intuition 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., 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, $r_{i,j+1}$ is deferred to $r^{'}_{i,j+1}$ in Figure 3, but it still has to complete 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 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. According to fixed priority scheduling theory, $r^{'}_{i,j+1}$ can be derived backwards from its deadline $d_{i,j+1}$ as follows:

$$r^{'}_{i,j+1} = d_{i,j+1} - \Theta_i(r^{'}_{i,j+1}, d_{i,j+1}) - C_i. \qquad (5)$$

Note that 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 in a recursive process from jobs of lower-priority transactions to higher-priority transactions. Nevertheless, it does not require that 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 completion of job $J_{i,j}$, deadline of its next job, $d_{i,j+1}$, is firstly derived 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 complete 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 completes and it is not needed by jobs of lower-priority transactions. 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,

$$LSD_i(t) = \max_{J_{i,j} \in S_J(t)} \{d_{i,j}\}\ (j \geq 0). \qquad (6)$$

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

$$LSD_i(t) \geq d_{k,j}, \qquad (7)$$

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 $J_{3,1}$, $J_{2,3}$, $J_{2,4}$, $J_{3,2}$ are shifted from times 14, 21, 28, 28 in *ML* to 18, 22, 30, 35 in *DS-FP*, respectively. □

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

3.2 *Deferrable Scheduling* Algorithm

This section presents the *DS-FP* algorithm. It is a *fixed priority* scheduling algorithm. Given an update transaction set $\mathcal{T}$, it is assumed that τ_i has higher priority than τ_j if

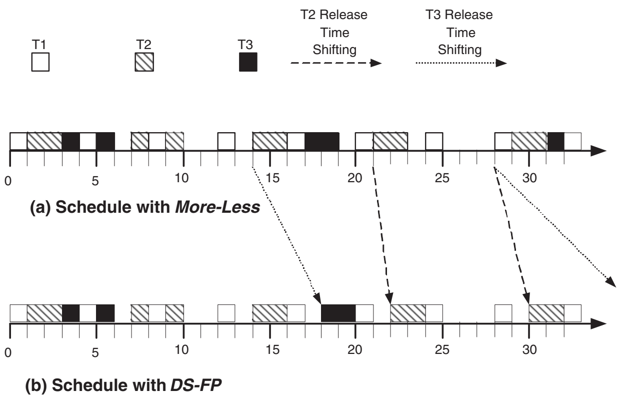


Figure 4. **Comparing *ML* and *DS-FP* schedules**

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**

$i < j$. Transaction priority assignment policy in *DS-FP* is the same as in *ML*, i.e., *Shortest Validity First*. 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). Then the sampling time of $J_{i,k+1}$, $r_{i,k+1}$, is derived backwards from its deadline by accounting for the interference from higher-priority transactions (Line 29).

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  $t \leftarrow 0$; // Initialization
3  // $LSD_i$ – Latest Scheduled Deadline of $\tau_i$'s jobs.
4  $LSD_i \leftarrow 0, \forall i \ (1 \leq i \leq m)$;
5  $\ell_i \leftarrow 0, \forall i \ (1 \leq i \leq m)$;
6  // $\ell_i$ is the latest scheduled job of $\tau_i$
7  for $i = 1$ to $m$ do
8      // Schedule finish time for $\tau_{i,0}$.
9      $r_{i,0} \leftarrow 0$;
10     $f_{i,0} \leftarrow C_i$;
11     // Calculate higher-priority (HP) preemptions.
12     oldHPPreempt $\leftarrow 0$;      // initial HP preemptions
13     hpPreempt $\leftarrow$ CalcHPPreempt$(i, 0, 0, f_{i,0})$;
14     while (hpPreempt > oldHPPreempt) do
15         // Accounting for the interference of HP tasks
16         $f_{i,0} \leftarrow r_{i,0}$ + hpPreempt + $C_i$;
17         if $(f_{i,0} > \mathcal{V}_i - C_i)$ then abort endif;
18         oldHPPreempt $\leftarrow$ hpPreempt;
19         hpPreempt $\leftarrow$ CalcHPPreempt$(i, 0, 0, f_{i,0})$;
20     end
21     $d_{i,0} \leftarrow f_{i,0}$;
22 end
23 return;
25 case (upon completion of $J_{i,k}$) :
26 // Schedule release time for $J_{i,k+1}$.
27 $d_{i,k+1} \leftarrow r_{i,k} + \mathcal{V}_i$;      // get next deadline for $J_{i,k+1}$
28 // $r_{i,k+1}$ is also the sampling time for $J_{i,k+1}$
29 $r_{i,k+1} \leftarrow$ ScheduleRT$(i, k+1, C_i, d_{i,k+1})$;
30 return;
```

Algorithm 3.2 **ScheduleRT***$(i,\ k,\ C_i,\ d_{i,k})$:*

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

Output: $r_{i,k}$.

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

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 interval $[r_{i,k},\ d_{i,k})$, and adjusts $r_{i,k}$ by accounting for interference from higher-priority transactions (Lines 5 to 12). The computation of $r_{i,k}$ continues until interference from higher-priority transactions does not change in an iteration. In particular, Line 9 detects an 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 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.

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  $\ell_i \leftarrow k$;            // Record latest scheduled job of $\tau_i$.
2  $d_{i,k} \leftarrow t_2$;
3  $LSD_i \leftarrow t_2$;
4  if $(i = 1)$
5    then  // No preemptions from higher-priority tasks.
6          return 0;
7  elsif $(LSD_{i-1} \ge t_2)$
8    then      // Get preemptions from $\tau_j$ $(\forall j, 1 \le j < i)$
9          // because $\tau_j$'s schedule is complete before $t_2$.
10         return GetHPPreempt$(i, k, t_1, t_2)$;
11 endif
12 // build $\mathcal{S}$ up to or exceeding $t_2$ for $\tau_j$ $(1 \le j < i)$.
13 for $j = 1$ to $i - 1$ do
14     while $(d_{j,\ell_j} < t_2)$ do
15       $d_{j,\ell_j+1} \leftarrow r_{j,\ell_j} + \mathcal{V}_j$;
16       $r_{j,\ell_j+1} \leftarrow$ ScheduleRT$(j, \ell_j + 1, C_j, d_{j,\ell_j+1})$;
17       $\ell_j \leftarrow \ell_j + 1$;
18       $LSD_j \leftarrow d_{j,\ell_j}$;
19     end
20 end
21 return GetHPPreempt$(i, k, t_1, t_2)$;
```

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 interval $[t_1,\ t_2)$. Line 7 indicates 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 schedules of its higher-priority transactions are complete before t_2. In this case, the function simply returns 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. The following lemma states an important property of *ScheduleRT* when it terminates.

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

	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

Proof: This can be proved by contradiction. Suppose that $LSD_l(t) > LSD_j(t)$ $(k \ge l \ge j)$ when *ScheduleRT*$(i,\ k,\ C_i,\ d_{i,k})$ terminates at t. Let $t_2 = LSD_l(t)$ at Line 14 in *CalcHPPreempt*. As $LSD_j(t) < t_2$ at the same line, *ScheduleRT* has not reached the point to terminte. This contradicts the assumption. □

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

Example 3.2: Table 3 presents comparison of (release time, deadline) pairs of τ_1, τ_2 and τ_3 jobs before time 40 in Example 3.1, which are assigned by *ML* and *DS-FP* (Algorithm 3.1). 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 Algorthim 3.1 invokes *ScheduleRT*$(3, 1, 2, 20)$ at Line 29, which will derive $r_{3,1}$. 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 out 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 $d_{2,3}$ has been built for transactions with priority 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 one similar to the *sporadic* task model [1]. However, 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* satisfies the following condition:

$$\mathcal{V}_i - C_i \geq r_{i,j} - r_{i,j-1} \geq \mathcal{V}_i - \mathit{WCRT}_i \quad (j \geq 1), \quad (8)$$

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 for the execution of job $J_{i,j}$ (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.

ML can be regarded as a special case of *DS-FP* in which sampling (or release) time $r^{ml}_{i,j+1}$ and deadline $d^{ml}_{i,j+1}$ $(j \geq 0)$ can be specified as follows:

$$\begin{aligned} d^{ml}_{i,j+1} &= r^{ml}_{i,j} + \mathcal{V}_i, & (9)\\ r^{ml}_{i,j+1} &= d^{ml}_{i,j+1} - (\Theta_i(r^{ml}_{i,0}, f^{ml}_{i,0}) + C_i). & (10) \end{aligned}$$

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

3.4 Theoretical Analysis of *DS-FP* Utilization

Given a transaction set $\mathcal{T}$ that can be scheduled by *ML*, we now present the analysis of average processor utilization of *DS-FP*. Suppose $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$, let $\overline{U}_{DS}$ denote the average processor utilization under *DS-FP*. We have the average relative deadline of τ_i, namely $\overline{D}_i$, as follows:

$$\overline{D}_i = \textstyle\sum_{j=1}^{i}(\overline{n}_{i,j} \times C_j) \ (1 \leq i \leq m)$$

where $\overline{n}_{i,j}$ denotes the average number of times that higher priority transaction τ_j occurs during an interval of length $\overline{D}_i$. Therefore, $\sum_{j=1}^{i}(\overline{n}_{i,j} \times C_j)$ represents the average response time of τ_i. It is obvious that for any i, $\overline{n}_{ii} = 1$ and $\overline{n}_{i,j} = \frac{\overline{D}_i}{\overline{P}_j}$ where $\overline{P}_j$ is the average period for τ_j. Thus,

$$\overline{D}_i = C_i + \sum_{j=1}^{i-1}[(\frac{\overline{D}_i}{\overline{P}_j}) \times C_j] \quad (1 \leq i \leq m). \quad (11)$$

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

$$P_{i,j} + D_{i,j+1} = \mathcal{V}_i. \quad (12)$$

Given the first n jobs of τ_i, we have

$$\sum_{j=0}^{n-1}(P_{i,j} + D_{i,j+1}) = n \cdot \mathcal{V}_i.$$

Therefore,

$$\frac{1}{n}(\sum_{j=0}^{n-1} P_{i,j}) + \frac{1}{n}(\sum_{j=0}^{n-1} D_{i,j+1}) = \mathcal{V}_i. \quad (13)$$

Given arbitrarily large n $(n \to \infty)$, the following equation holds:

$$\overline{P}_i + \overline{D}_i = \mathcal{V}_i. \quad (14)$$

Following Eq. 11 and 14, $\overline{D}_i$ 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:

$$\begin{aligned} \overline{D}_i &= \frac{C_i}{1 - \sum_{j=1}^{i-1}\frac{C_j}{\overline{P}_j}} \quad (1 \leq i \leq m) & (15)\\ \overline{P}_i &= \mathcal{V}_i - \overline{D}_i \quad (1 \leq i \leq m) & (16) \end{aligned}$$

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

$$\overline{U}_{DS} = \sum_{i=1}^{m}\frac{C_i}{\overline{P}_i} = \sum_{i=1}^{m}(\frac{C_i}{\mathcal{V}_i - \frac{C_i}{1 - \sum_{j=1}^{i-1}\frac{C_j}{\overline{P}_j}}}) \quad (17)$$

The following example illustrates how the average utilization is estimated.

Example 3.3: Given the transaction set in Table 2, we calculate the average relative deadline and period of τ_i $(i = 1, 2, 3)$ as follows:

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

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%). □

4 Performance Evaluation

This section presents important results from our experimental studies of the proposed deferrable scheduling algorithm (*DS-FP*). We compare *DS-FP* with the More-Less (*ML*) algorithm, which outperforms Half-Half [22].

4.1 Simulation Model and Parameters

We have conducted two sets of experiments comparing the performance of *DS-FP* and *ML*. In the first set of experiments, we compare the update transaction workloads produced by *DS-FP* and *ML*. It is demonstrated that *DS-FP*

Parameter Class	Parameters	Meaning
System	N_{CPU}	No. of CPU
	N_T	No. of real-time data objects
	$\mathcal{V}_i$	Validity interval of X_i
Update Transactions	C_i	CPU time for updating X_i
	Length	No. of data to update
Triggered Transactions	CPU Time	CPU time per data access
	Length	No. of data to access
	Arrival Rate	Transaction triggering rate
	Slack Factor	Transaction slack factor

Table 4. Experimental parameters

Parameter Class	Parameters	Meaning
System	N_{CPU}	1
	N_T	[50, 300]
	$V_i(ms)$	[4000, 8000]
Update Transactions	$C_i(ms)$	[5, 15]
	Length	1
Triggered Transactions	CPU Time (ms)	[3, 5]
	Length	[5, 15]
	Arrival Rate	[5, 10]
	Slack Factor	8

Table 5. Experimental settings

produces lower CPU workload than *ML*. Also, we demonstrate that the increase of average sampling period from *DS-FP* is the main reason for its lower workload. In the second set of experiments, we study the performance of *DS-FP* and *ML* under mixed transaction workloads: a class of update transactions that maintain the freshness (validity) of real-time data objects, and a class of triggered transactions that are triggered by the changes of real-time data values. The triggered transactions need to read a group of real-time data objects for decision making. Given transactions belonging to different classes, update transactions are assigned higher priorities than the triggered transactions. For simplicity of the simulation study, only one version of a real-time data object is maintained. Upon refreshing a real-time data object, the older version is discarded. The primary performance metrics used in the experiments are CPU workload, mean transaction response time and missed deadline ratio (*MDR*) of the triggered transactions. We also measure the age of data (*AGE*) that indicates how old the real-time data object is at the commit time of a triggered transaction. Suppose that the current data value for real-time data object X_i is sampled at time t, and its data value is valid until $t + \mathcal{V}_i$. If a triggered transaction that reads the value of X_i commits at time t', its *AGE* is defined as:

$$AGE(X_i,\, t') = \begin{cases} \frac{t'-t}{\mathcal{V}_i} & : \quad t' > t + \mathcal{V}_i \\ 1 & : \quad t' \leq t + \mathcal{V}_i \end{cases}$$

A summary of the parameters and default settings used in experiments are presented in Tables 4 and 5. The baseline values for the parameters follow those used in [22], which are originally from air traffic control applications. Three classes of parameters are defined: system parameters, update transaction parameters and triggered transaction parameters. For system configurations, we only consider a single CPU, main memory based RTDBS. The number of real-time data objects is varied from 50 to 300 and it is assumed that the validity interval length of each real-time data object is uniformly varied from 4000 to 8000 ms. For update transactions, it is assumed that each transaction updates one real-time data object, and the CPU time for each transaction is uniformly varied from 5 to 15 ms. For the triggered transactions, we assume that the number of real-time data objects accessed by each transaction is uniformly varied from 5 to 15, while accessing each data object takes 3 to 5 ms of CPU time. The inter-arrival time of the triggered transactions follows exponential distribution and the arrival rate is varied from 5 to 10 transactions per second. The slack factor determines the slack of a transaction before its deadline expires and it is fixed at 8. Let $AT(\tau_i)$, $ET(\tau_i)$ and $Deadline(\tau_i)$ denote the arrival time, total execution time and deadline of triggered transaction τ_i, the deadline of τ_i can be calculated as follows in our experiments:

$$Deadline(\tau_i) = AT(\tau_i) + (ET(\tau_i) \times Slack\ Factor)$$

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.

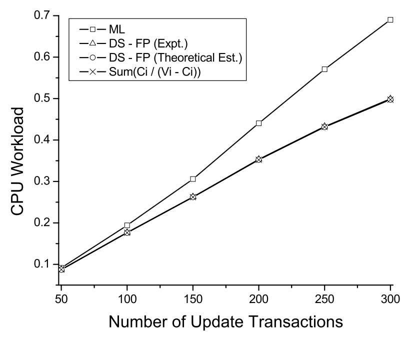


Figure 5. CPU workloads comparison

4.2 Expt. 1: Comparison of CPU Workloads

In this set of 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 5.

The resulting CPU workloads generated from *ML* and *DS-FP* are depicted in Figure 5. From the results, we observe that *DS-FP*'s CPU workload is consistently lower than that of *ML*. In fact, the difference widens as the number of update transactions increases. The difference reaches 18%

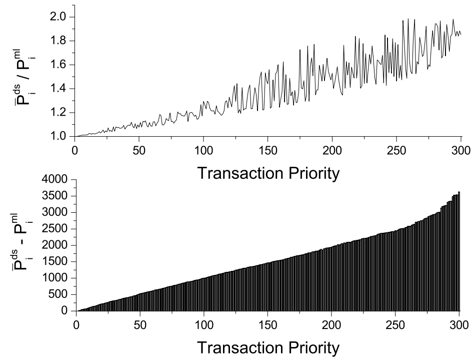


Figure 6. **Average sampling period comparison**

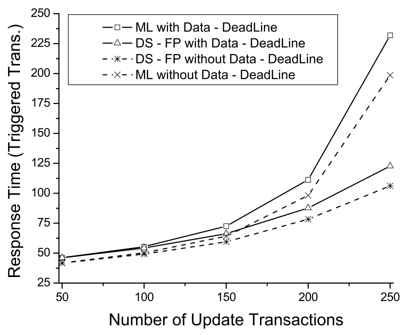


Figure 7. **Response time comparison**

when the number of transactions is 300. It is also observed that our experimental results of *DS-FP* match the average CPU workload calculated from the theoretical estimation (Eq. 17). Last but not least, 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 5. The improvement of CPU workload under *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 6 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 [22], 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.

$$\overline{P}_i^{ds} = \frac{1}{n-1} \sum_{j=1}^{n-1} (r_{i,j} - r_{i,j-1}) \tag{18}$$

In Figure 6, 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 decrease of the transaction's priority. *DS-FP* reduces the average sampling rate more for lower-priority transactions, thus greatly reduces the workload of CPU. Figure 6 also demonstrates that the trend of $(\frac{\overline{P}_i^{ds}}{P_i^{ml}})$ increases similarly although it fluctuates.

In summary, when a set of update transactions is scheduled by *DS-FP* to maintain temporal validity of real-time data objects, it produces a schedule with a much lower CPU workload than *ML*. Thus more CPU capacity is available for other transactions, e.g., triggered transactions.

4.3 Expt. 2: Co-scheduling of Mixed Workloads

In this set of experiments, performances of mixed transactions are compared for *ML* and *DS-FP* in two scenarios: 1) Triggered transactions do not have deadlines. Their average response time and age of data at commit time are compared for *ML* and *DS-FP*; 2) Triggered transactions have deadlines. Their missed deadline ratios (MDRs) are compared for *ML* and *DS-FP*. In both cases, update transactions are scheduled by either *ML* or *DS-FP* for comparison. The parameter settings in the experiments are listed in Table 5.

4.3.1 Comparison of Average Response Time

In these experiments, triggered transactions do not have deadlines. We fix the triggered transactions' arrival rate at 10 transactions per second and the CPU time for accessing each real-time data object at 3 ms. We vary the number of update transactions from 50 to 250 in order to show its impact on the performance of average response times of the triggered transactions. We also consider two cases: 1) Triggered transactions obey *data-deadline* [21], which means that triggered transactions are aborted and restarted if the value of a real-time data object accessed by the transaction expires before the transaction commits; 2) Triggered transactions do not obey *data-deadline*, which means that the triggered transactions can still commit if the values of its accessed real-time data objects expire. Informally, data-deadline is a deadline assigned to a transaction due to the temporal constraints (i.e., validity interval length) of the data accessed by the transaction. For details of the concept of data-deadline, readers are referred to [21].

In Figure 7, the average response time of triggered transactions with update transactions scheduled by *DS-FP* is consistently lower than that of triggered transactions with update transactions scheduled by *ML*. For example, there is a 20% improvement in the response time of triggered transactions if the number of update transactions is 200 no matter whether *data-deadline* is obeyed or not. Figure 7 also demonstrates that the average response time of the triggered transactions in *ML* increases dramatically when the number

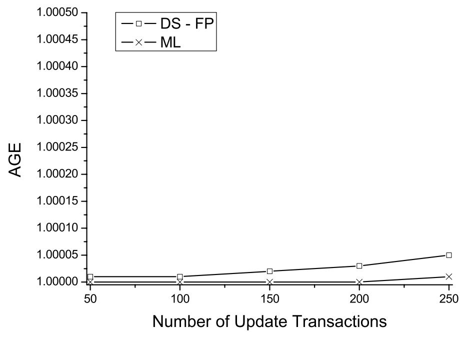


Figure 8. **Average age of data**

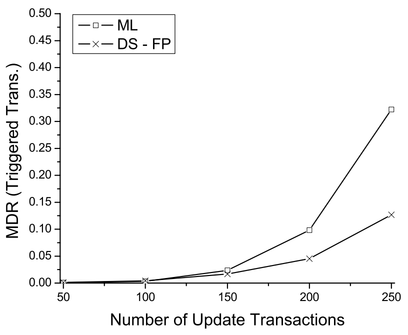


Figure 9. **Missed deadline ratio comparison**

of update transactions reaches 250. This is because the CPU is almost saturated by the workload generated from update transactions in the *ML* case if the number of update transactions reaches 250. However, the CPU workload of update transactions generated from *DS-FP* is much lower than that from *ML*.

4.3.2 Comparison of Average Age of Data

In these experiments, triggered transactions again do not have deadlines, and *data-deadline* is not obeyed by the triggered transactions. We compare the average age of data (AGE) accessed by the triggered transactions for *ML* and *DS-FP* at transaction commit time. Because *DS-FP* samples the data at lower rate, it is unclear how much impact the lower sampling rate has on the age of data at the commit time of a triggered transaction. Figure 8 demonstrates that *DS-FP*'s average age of data at commit time of the triggered transactions is only slightly larger than that of *ML*. In fact, the difference is very small and it can be totally ignored.

4.3.3 Comparison of Missed Deadline Ratio (MDR)

Different from previous experiments, in this set of experiments we suppose that the triggered transactions have deadlines and they obey the *data-deadline* constraint. A triggered transaction that cannot commit before the validity of its accessed data expires has to be aborted, and restarted later if it has not missed its deadline. In such a case, a data-deadline is imposed on the triggered transaction due to the temporal constraints resulting from data validities. The triggered transactions are scheduled by the earliest deadline first (EDF) scheduling algorithm [14]. The CPU time for accessing a real-time data object is fixed at 5 ms. Figure 9 shows that the MDR of the triggered transactions under *DS-FP* is much lower than that under *ML*. For example, when the number of update transactions is 200 and the triggered transactions are scheduled by EDF, only 4% triggered transactions miss their deadlines under *DS-FP*, but around 10% triggered transactions miss their deadlines under *ML*. This is because the CPU workload of update transactions under *ML* is much higher than that under *DS-FP*. It can be observed that the difference of MDRs of triggered transactions widens as the number of update transactions increases.

In summary, *DS-FP* also provides better performance in the co-scheduling of mixed workloads where transactions can be triggered by the changes of values of real-time data objects. It greatly improves the average transaction response time and missed deadline ratio while only increases the average age of data insignificantly.

5 Related Work

There has been a lot of work on RTDBSs in which validity intervals are associated with real-time data [19, 9, 10, 11, 4, 21, 8, 12, 6, 5, 22]. [6] presents a vehicular application with embedded engine control systems, and an on-demand scheduling algorithm for enforcing base and derived data freshness. [5] proposes an algorithm (ODTB) for updating data items that can skip unnecessary updates allowing for better utilization of the CPU in the vehicular application. In [11], similarity-based principles are coupled with the *Half-Half* approach to adjust real-time transaction load by skipping the execution of task instances. The concept of *data-deadline* is proposed in [21]. It also proposes data-deadline based scheduling, forced-wait and similarity-based scheduling techniques to maintain the temporal validity of real-time data and meet transaction deadlines in RTDBSs.

Our work is related to the *More-Less* scheme in [22, 12]. *More-Less* 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 sporadic task model based *DS-FP* algorithm in which deadline of a transaction job is derived adaptively, and the separation of two consecutive jobs is not a constant. *DS-FP* reduces 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

[7]. The *EDL* algorithm proposed in [3] processes tasks as late as possible based on the Earliest Deadline scheduling algorithm [14]. *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 [2, 16] to reduce processor utilization.

6 Conclusions and Future Work

This paper proposed a novel algorithm – deferrable scheduling for fixed priority transactions (*DS-FP*). Distinct from past studies of maintaining freshness (or temporal validity) of data in which the periodic task model is adopted, *DS-FP* adopts the *sporadic* task model. The deadlines of jobs and separation of two consecutive jobs of an update transaction are adjusted judiciously so that the farthest distance of the sampling time of a job and the completion time of its next job is bounded by the validity length of the updated real-time data. We proposed 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 processor workload compared to *ML*. Thus, *DS-FP* can improve the performance of triggered transactions when it is used by a RTDBS to track environmental changes.

We intend to investigate the feasibility of *DS-FP* in our future work. For example, it is neither clear what a sufficient and necessary condition is for feasibility of *DS-FP*, nor is it clear how much *DS-FP* can improve the feasibility of update transactions compared to *ML*. Moreover, the concept of *deferrable scheduling* is only used to schedule update transactions with fixed priority in this paper. 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 [14, 3] of update transactions.

Acknowledgement
This work was partially supported by the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1076/02E].

References

[1] S. K. Baruah, A. K. Mok, L. E. Rosier, "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor," *IEEE Real-Time Systems Symposium*, December 1990.

[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] 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.

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

[5] 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.

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

[7] 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.

[8] 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.

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

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

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

[12] K.Y. Lam, M. Xiong, B. Liang and Y. Guo, "Statistical Quality of Service Guarantee for Temporal Consistency of Real-time Data Objects", *IEEE Real-Time Systems Symposium*, 2004.

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

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

[15] 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.

[16] L. Lundberg, "Utilization Based Schedulability Bounds for Age Constraint Process Sets in Real-Time Systems," *Real-Time Systems*, 23(3), pp. 273-295, 2002.

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

[18] 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.

[19] 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.

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

[21] 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.

[22] M. Xiong and K. Ramamritham, "Deriving Deadlines and Periods for Real-Time Update Transactions," *IEEE Transactions on Computers*, 53(5), pp. 567-583, 2004.