Deriving Deadlines and Periods for Real-Time Update Transactions *

Ming Xiong†

Bell Laboratories

600 Mountain Avenue

Murray Hill, NJ 07974

Email: mxiong@bell-labs.com

Krithi Ramamritham‡

Department of Computer Science and Engineering

Indian Institute of Technology Bombay

Mumbai, India 400076

Email: krithi@cse.iitb.ac.in

Abstract

Typically, temporal validity of real-time data is maintained by periodic update transactions. In this paper, we examine the problem of period and deadline assignment for these update transactions such that (1) these transactions can be guaranteed to complete by their deadlines and (2) the imposed CPU workload is minimized. To this end, we propose a novel approach, named the More-Less approach. By applying this approach, updates occur with a period which is more than the period obtained through traditional approaches but with a deadline which is less than the traditional period. We show that the More-Less approach is better than existing approaches in terms of schedulability and the imposed load. We examine the issue of determining the assignment order in which transactions must be considered for period and deadline assignment so that the resulting CPU workloads can be minimized. To this end, the More-Less approach is first examined in a restricted case where the Shortest Validity First (SVF) order is shown to be an optimal solution. We then relax some of the restrictions and show that SVF is an approximate solution which results in CPU workloads that are close to the optimal solution. Our analysis and experiments show that the More-Less approach is an effective design approach that can provide better schedulability and reduce update transaction CPU workload while guaranteeing data validity constraints.

Index Terms - Real-time database systems, temporal validity, deadline monotonic scheduling, fixed priority scheduling, real-time transaction processing.

---

* A preliminary version of this paper [15] appeared in the 20th IEEE Real-Time Systems Symposium, Phoenix, Arizona, December, 1999.

† Work was partially done while the authors were affiliated with University of Massachusetts, Amherst.

‡ Contact author. Research supported in part by the National Science Foundation Grant IRI-9619588 and CDA-9502639. Krithi Ramamritham is also affiliated with University of Massachusetts, Amherst.

1 Introduction

A real-time database (RTDB) is composed of real-time objects which are updated by periodic sensor transactions. An *object* in the database models a real world entity, for example, the position of an aircraft. A real-time object is one whose state may become invalid with the passage of time. Associated with the state is a temporal validity interval. To monitor the states of objects faithfully, a real-time object must be refreshed by a sensor transaction before it becomes invalid, i.e., before its temporal validity interval expires. The actual length of the temporal validity interval of a real-time object is application dependent. Sensor transactions are generated by intelligent sensors which periodically sample the value of real-time objects. When sensor transactions arrive at RTDBs with sampled data values, their updates are installed and real-time data are refreshed. So one of the important design goals of RTDBs is to guarantee that temporal data remain fresh, i.e., they are always valid. RTDBs that do not keep track of temporal validity of data may not be able to react to abnormal situations in time. Therefore, efficient design approaches are desired to guarantee the freshness of temporal data in RTDBs while minimizing the CPU workload resulting from periodic sensor transactions.

In this paper, we propose the *More-Less* approach, a design approach which maintains the freshness of temporal data while reducing the CPU workload incurred by periodic sensor transactions. In general, *More-Less* can be applied to multi-processor systems. In this paper, however, we focus our study on single processor systems. It is shown that the *More-Less* approach outperforms traditional approaches in terms of sensor transaction schedulability and imposed CPU workload. Using the *More-Less* approach, transactions are considered in a given priority order and their periods and deadlines are assigned. So an important issue is to determine the priority order so that the CPU workload imposed by transactions can be minimized. It is demonstrated, through both analysis and experiments, that *Shortest Validity First (SVF)* is an efficient assignment order to minimize CPU workload for update transactions.

This paper is organized as follows: Section 2 reviews traditional approaches and introduces the intuition underlying the *More-Less* approach. The *More-Less* approach is formally introduced in Section 3, and compared with a traditional approach. We also examine the issue of determining the assignment order. Specifically, we propose and analyze *SVF*, an efficient transaction assignment order to minimize CPU workload. An application of the *More-Less* approach is discussed in Section 4. Experimental results are presented in Section 5. We discuss related work in Section 6, and conclude the paper in Section 7.

2 Design Approaches

In this section, traditional approaches for maintaining temporal validity, namely the *One-One* and *Half-Half* approaches, are reviewed. The *More-Less* approach is then introduced through an example.

From here on, $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ refers to a set of periodic sensor transactions $\{\tau_1, \tau_2, .., \tau_m\}$ and $\mathcal{X} = \{X_i\}_{i=1}^{m}$ refers to a set of temporal data. All temporal data are assumed to be kept in main memory. Associated with X_i $(1 \leq i \leq m)$ is a validity interval of length α_i: transaction τ_i $(1 \leq i \leq m)$ updates the corresponding data X_i. Because each sensor transaction updates different data, no concurrency control is considered for sensor transactions. We assume that a sensor always samples the value of a temporal data at the beginning of its period, and all the first instances of sensor transactions are initiated at the

Symbol	Definition
X_i	Temporal Data
τ_i	Periodic sensor transaction updating X_i
I_i	Release time of the first instance of τ_i
J_{ij}	The jth instance of τ_i
R_{ij}	Response time of the jth instance of τ_i
C_i	Computation time of transaction τ_i
α_i	Validity interval length of X_i
L_i	Validity interval slack of transaction τ_i, i.e., $L_i = \alpha_i - C_i$.
P_i	Period of transaction τ_i
D_i	Relative deadline of transaction τ_i
δ_i	Jitter (or arrival latency) bound of transaction τ_i
$\tau_i \rightarrow \tau_j$	In an assignment order, transaction τ_i precedes transaction τ_j.
$W_i(t)$	The cumulative demands on the processor made by transactions $\tau_1, \tau_2, ..., \tau_i$ over time $[0, t]$, i.e., $W_i(t) = \sum_{j=1}^{i} C_j \lceil \frac{t}{P_j} \rceil$.
M_i	The maximum laxity for transaction τ_i over time $[0, D_i]$.
n_{ij}	the number of times τ_j occurs before the first instance of τ_i completes.
U_{ij}	Given an assignment order $\tau_i \rightarrow \tau_j$ of two adjacent transactions τ_i and τ_j, CPU workload of τ_i and τ_j, i.e., $U_{ij} = \frac{C_i}{P_i} + \frac{C_j}{P_j}$

Table 1. Symbols and definitions.

same time. However, a sensor transaction generated by that sensor may arrive at a RTDB with an arrival latency, which is also referred to as jitter. The jitter of transaction τ_i is defined as follows:

Jitter of τ_i = Arrival Time of τ_i - Sampling Time of τ_i.

where arrival time of τ_i is the time when τ_i arrives the RTDB, and sampling time of τ_i is the time when the value of temporal data X_i is sampled. We assume that the jitter of transaction τ_i is bounded by δ_i. The jitter of each transaction is zero unless specified otherwise. C_i, D_i and P_i $(1 \leq i \leq m)$ denote the execution time, relative deadline, and period of transaction τ_i, respectively. D_i of sensor transaction τ_i is defined as follows:

D_i = Deadline of τ_i - Sampling Time of τ_i.

Deadlines of sensor transactions are firm deadlines. Formal definitions of some of the often-used symbols are given in Table 1. Our goal is to determine P_i and D_i such that all the sensor transactions are schedulable and CPU workload resulting from sensor transactions is minimized.

We assume a simple execution semantics for periodic transactions: a transaction must be executed once every period.

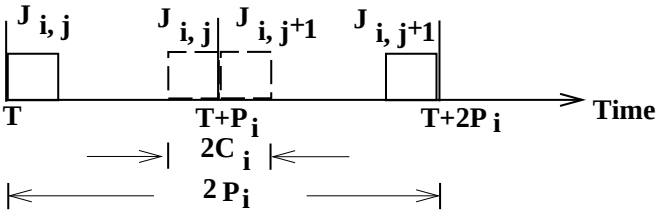

Figure 1. Extreme execution cases of periodic sensor transactions

However, there is no guarantee on when an instance of a periodic transaction is actually executed within a period. We also assume that these periodic transactions are preemptable.

2.1 One-One Approach

We introduce the first approach, in which the period and relative deadline of a sensor transaction have to be equal to the data validity interval. Because the separation of the execution of two consecutive instances of a transaction can exceed the validity interval, data can become invalid under the *One-One* approach. So this approach cannot guarantee the freshness of temporal data in RTDBs.

Example 2.1: Consider Figure 1: A periodic sensor transaction τ_i with deterministic execution time C_i refreshes temporal data X_i with validity interval α_i. The period P_i and relative deadline D_i of τ_i are assigned the value α_i. Suppose $J_{i,j}$ and $J_{i,j+1}$ are two consecutive instances of sensor transaction τ_i. Transaction instance $J_{i,j}$ samples data X_i with validity interval $[T, T+\alpha_i)$ at time T, and $J_{i,j+1}$ samples data X_i with validity interval $[T+\alpha_i, T+2\alpha_i)$ at time $T+\alpha_i$. From Figure 1, the actual arrival time of $J_{i,j}$ and finishing time of $J_{i,j+1}$ can be as close as $2C_i$, and as far as $2P_i$, i.e., $2\alpha_i$ when the period of τ_i is α_i. In the latter case, the validity of data X_i refreshed by $J_{i,j}$ expires after time $T+\alpha_i$. Since $J_{i,j+1}$ cannot refresh data X_i before time $T+\alpha_i$, X_i is invalid from $T+\alpha_i$ until it is refreshed by $J_{i,j+1}$, just before the next deadline $T+2\alpha_i$. □

2.2 Half-Half Approach

In order to guarantee the freshness of temporal data in RTDBs, the period and relative deadline of a sensor transaction are each typically set to be less than or equal to one-half of the data validity interval [11, 6]. In Figure 1, the farthest distance (based on the arrival time of a periodic transaction instance and the finishing time of its next instance) of two consecutive sensor transactions is $2P_i$. If $2P_i \leq \alpha_i$, then the freshness of temporal data X_i is guaranteed as long as instances of sensor transaction τ_i do not miss their deadlines.

Unfortunately, even though data freshness is guaranteed, this design approach at least doubles CPU workload of the sensor transaction in the RTDBs compared to the *One-One* approach. Next, we introduce a new approach which guarantees the freshness of temporal data but incurs much less CPU workload compared to the *Half-Half* approach.

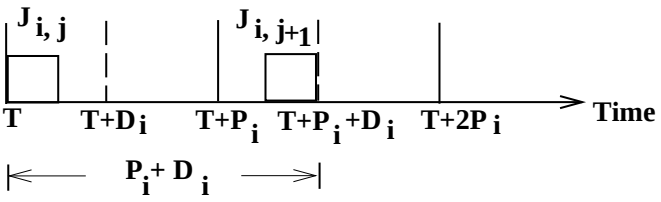

Figure 2. Illustration of *More-Less* approach

Approach	D_i	P_i	**Workload**
One-One	α_i	α_i	$U_o = \frac{C_i}{\alpha_i} = \frac{1}{5}$
Half-Half	$\frac{\alpha_i}{2}$	$\frac{\alpha_i}{2}$	$U_h = \frac{2C_i}{\alpha_i} = \frac{2}{5}$
More-Less	$\frac{\alpha_i}{3}$	$\frac{2\alpha_i}{3}$	$U_m = \frac{3C_i}{2\alpha_i} = \frac{3}{10}$

Table 2. Comparison of three approaches

2.3 *More-Less* Approach: Intuition

The goal of the *More-Less* approach is to minimize the CPU workload of sensor transactions while guaranteeing the freshness of temporal data in RTDBs. Recall that, for simplicity of discussion, we have assumed that a sensor transaction is responsible for updating a single temporal data item in the system. In *More-Less*, the period of a sensor transaction is assigned to be *more* than *half* of the validity interval of the temporal data updated by the transaction, while its corresponding relative deadline is assigned to be *less* than *half* of the validity interval of the same data. However, the sum of the period and relative deadline always equals the length of the validity interval of the data updated. Consider Figure 2. Let $P_i > \frac{\alpha_i}{2}$, $C_i \leq D_i < P_i$ where $P_i + D_i = \alpha_i$. The farthest distance (based on the arrival time of a periodic transaction instance and the finishing time of its next instance) of two consecutive sensor transactions J_{ij} and J_{ij+1} is $P_i + D_i$. In this case, the freshness of X_i can always be maintained if sensor transactions make their deadlines. Obviously, the load incurred by sensor transaction τ_i can be reduced if P_i is enlarged (which implies that D_i is shrunk.). Therefore, we have the constraints $C_i \leq D_i < P_i$ and $P_i + D_i = \alpha_i$ which aim at minimizing the CPU workload of periodic transaction τ_i.

Example 2.2: Suppose there is temporal data X_i with validity interval α_i in a uniprocessor RTDB system. τ_i updates X_i periodically. For simplicity of discussion, we assume that $\delta_i = 0$. Our goal is to assign proper values to P_i and D_i given C_i and α_i so as to reduce the CPU workload resulting from sensor transaction τ_i. Suppose $C_i = \frac{1}{5}\alpha_i$, possible values of P_i, D_i and the corresponding CPU workload according to the three different design approaches are shown in Table 2. □

Only *Half-Half* and *More-Less* can guarantee the freshness of temporal data X_i if all the sensor transactions complete before their deadlines. We also notice that $U_o < U_m < U_h$ (see Table 2). If $P_i = \frac{N-1}{N}\alpha_i$, then $D_i = \frac{1}{N}\alpha_i$, where $N \geq 2$. The freshness of temporal data in RTDBs is guaranteed if all sensor transactions complete before their deadlines. In such a case, we also notice that $U_m = \frac{NC_i}{(N-1)\alpha_i}$ and $U_o \leq U_m < U_h$. Theoretically, if $N \to \infty$, $U_m \to U_o$.

Unfortunately, how close U_m can get to U_o depends on C_i since $D_i \geq C_i$ implies $\frac{\alpha_i}{C_i} \geq N$. As N increases, relative deadlines become shorter and sensor transactions are executed with more stringent time constraints.

Therefore, given a set of sensor transactions in RTDBs, we need to find periods and deadlines of update transactions based on the temporal validity intervals of data such that the CPU workload of sensor transactions is minimized and the schedulability of the resulting sensor transactions is guaranteed. The *More-Less* approach achieves this, as shown in the next section.

3 *More-Less*: Analysis and Results

In this section, we formally introduce the *More-Less* approach with three constraints: the *Validity Constraint*, the *Deadline Constraint* and the *Schedulability Constraint*. We then show that the schedulability of transactions and data freshness are guaranteed by *More-Less*. Next, the impact of jitter on *Half-Half* and *More-Less* is discussed. To understand the advantages of *More-Less*, we then compare *More-Less* with *Half-Half* and show that *More-Less* is superior to *Half-Half* in terms of schedulability and for minimizing CPU workload. We show that the *assignment order*, i.e., the order in which periods and deadlines are assigned has an important impact on schedulability and CPU workload of solutions derived from *More-Less*. Therefore, to find an optimal assignment order for *More-Less*, we investigate the issues of assignment order with the aid of a concept named *partitioning*. We show that *Shortest Validity First (SVF)*, an assignment order proposed in this paper, results in an optimal solution under certain restrictions. With the relaxation of some of the restrictions, it is proved that SVF produces an approximate solution within a certain bounded range of optimal solutions in general. SVF is shown to be a good heuristic solution in many applications, especially, where *validity interval lengths are much larger than transaction computation times*.

3.1 The Design Approach

Although dynamic-priority scheduling is in general more effective than fixed-priority scheduling, it is also more difficult to implement and hence can incur higher system overhead than fixed-priority scheduling. Moreover, for many applications, it is possible to implement fixed-priority algorithms at the hardware level by the use of a priority-interrupt mechanism. Thus, the overhead involved in scheduling tasks can be reduced to a minimal level [10]. Given this, we study fixed-priority scheduling algorithms in this paper. Addressing this problem under dynamic-priority scheduling is left as future work.

In our previous discussions of Example 2.2, update sensor transactions are assumed to sample data at the beginning of their periods, and those transactions arrive at the system without latency. That is, it is assumed that jitter is constrained to be zero for update transactions to be available in the system. We relax this assumption here and discuss the impact of jitter on the *Half-Half* and *More-Less* approach.

In real-time systems, arrival jitter can usually be bounded. We consider two cases for arrival jitter of update transactions. In the first case, the arrival jitter of any update transactions is bounded by $\delta \geq 0$, i.e., the maximum arrival jitter of each update transaction is a constant δ. In the second case, each update transaction has its own arrival jitter bound, e.g., the maximum jitter of transaction τ_i $(1 \leq i \leq m)$ is a value $\delta_i \geq 0$. It should be noted that a jitter bound of each transaction should not be larger than its deadline. Otherwise, the transaction is not schedulable. We first investigate transactions with the same arrival

jitter bound because this is the simpler case to analyze. We then relax it and discuss the case in which transactions have arbitrary jitter bounds.

Before *More-Less* is presented, some theoretical background for periodic tasks is reviewed in Section 3.1.1. It should be noted that theorems relating to periodic tasks presented in Section 3.1.1 also hold for periodic sensor transactions.

3.1.1 Theoretical Background for Periodic Tasks

First, consider the longest response time for any instance of a periodic task τ_i where the response time is the difference between the task initiation time $(I_i + KP_i)$ $(K = 0, 1, 2, ...)$ and the task completion time where I_i is the release time of the first instance of τ_i.

Lemma 3.1: For a set of periodic tasks $\mathcal{T} = \{\tau_i\}_{i=1}^m$ with task initiation time $(I_i + KP_i)$ $(K = 0, 1, 2, ...)$, the longest response time for any instance of τ_i occurs for the first instance of τ_i when $I_1 = I_2 = ... = I_m = 0$. [7]

For $I_i = 0$ $(1 \leq i \leq m)$, the tasks are *in phase* because the first instances of all the tasks are initiated at the same time. It should be noted that we only discuss *in phase* tasks in this paper. A time instant after which a task has the longest response time is called a *critical instant*, e.g., time 0 is a critical instant for all the tasks if those tasks are *in phase*.

Further, Leung and Whitehead [10] introduced a fixed-priority scheduling algorithm, the *deadline monotonic* scheduling algorithm, in which task priorities are assigned inversely with respect to task deadlines, that is, τ_i has higher priority than τ_j if $D_i < D_j$ [10].

Lemma 3.2: For a set of periodic tasks $\mathcal{T} = \{\tau_i\}_{i=1}^m$ with $D_i \leq P_i$ $(1 \leq i \leq m)$ and $I_i = 0$ $(1 \leq i \leq m)$, the *deadline monotonic* scheduling algorithm is an optimal fixed priority scheduling algorithm. A task set is schedulable by this algorithm if the first instance of each task after a *critical instant* meets its deadline.

Since the *deadline monotonic* algorithm is an optimal fixed priority scheduling algorithm for a set of tasks $\{\tau_i\}_{i=1}^m$ with $D_i \leq P_i$ $(1 \leq i \leq m)$, it is used to maintain the schedulability of periodic transactions in our approach.

3.1.2 *More-Less* Definitions

More-Less is formally defined as follows.

Definition 3.1: **More-Less**: For a given set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^m$, the *More-Less* approach determines deadlines and periods of transactions such that the following three constraints are satisfied:

- *Validity Constraint:* $P_i + D_i \leq \alpha_i$
- *Deadline Constraint:* $\delta + C_i \leq D_i \leq P_i$
- *Schedulability Constraint:* Without loss of generality, assume that for $i < j$, $\tau_i \rightarrow \tau_j$ (i.e., τ_i precedes τ_j when they are considered for deadline and period assignment[1]). Because the transactions are scheduled by the *deadline monotonic*

[1] We use assignment order and priority order interchangeably in this paper. For example, $\tau_i \rightarrow \tau_j$ also means τ_i has higher priority than τ_j.

algorithm, the following inequality constraint must hold:

$\delta + \sum_{j=1}^{i}(n_{ij} \times C_j) \leq D_i \ (1 \leq i \leq m)$,

where δ is transaction jitter bound and n_{ij} denotes the number of times transaction τ_j occurs before the first instance of τ_i completes. Therefore, $\sum_{j=1}^{i}(n_{ij} \times C_j)$ represents the response time of the first instance of τ_i. It is easy to see that for any i, $n_{ii} = 1$.

The next theorem proves that data freshness can be guaranteed under *More-Less*.

Theorem 3.1: Given a set of update transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ $(m \geq 1)$ with deadlines and periods determined by *More-Less*, the set of transactions is schedulable and data freshness is guaranteed.

Proof: We need to prove that the three constraints of the *More-Less* approach can guarantee the schedulability of transactions and freshness of data. Because of the *schedulability constraint*, the first instance of every transaction can meet its deadline. Combined with the *deadline constraint*, it follows from Lemma 3.2 that the set of transactions can be scheduled by the *deadline monotonic* algorithm. Since transactions satisfy the *validity constraint*, data freshness can also be guaranteed. Hence the set of transactions is schedulable and data freshness is guaranteed. □

Given the *More-Less* approach, the optimization problem we need to solve is a non-linear programming problem: Determine D_i and P_i such that

$$U_m = \sum_{i=1}^{m} \frac{C_i}{P_i}$$

is minimized subject to the three constraints above.

From the three constraints underlying *More-Less*, we know that $P_i \leq \alpha_i - \sum_{j=1}^{i}(n_{ij} \times C_j) - \delta$. Let $P_i = \alpha_i - \sum_{j=1}^{i}(n_{ij} \times C_j) - \beta_i$ $(\beta_i \geq \delta)$. Now we transform the problem to an assignment order problem so that $U_m = \sum_{i=1}^{m} \frac{C_i}{\alpha_i - \sum_{j=1}^{i}(n_{ij} \times C_j) - \beta_i}$ is minimized, where $\beta_i \geq \delta$.

It is easy to see that if U_m is minimized, then $\beta_i = \delta$ for all i $(1 \leq i \leq m)$ and $D_i = \delta + \sum_{j=1}^{i}(n_{ij} \times C_j)$ $(1 \leq i \leq m)$. Now we have

$$P_i = \alpha_i - \delta - \sum_{j=1}^{i}(n_{ij} \times C_j) \ \ (1 \leq i \leq m). \tag{1}$$

In particular, if $D_i = P_i$ is true for all i $(1 \leq i \leq m)$, the *More-Less* approach actually reduces to the *Half-Half* approach. However, if there is at least one transaction τ_i $(1 \leq i \leq m)$ with $D_i < P_i$ from *More-Less*, it is referred to as *strict More-Less*.

Definition 3.2: Strict More-Less: For a given set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$, the *strict More-Less* approach determines deadlines and periods of transactions such that the validity constraint, deadline constraint, as well as schedulability constraint in *More-Less* can be satisfied, and $\exists k, 1 \leq k \leq m$, $D_k < P_k$.

Sets of transactions that are schedulable by *strict More-Less* or *Half-Half* are also schedulable by *More-Less*. We will show later in Section 3.4, there exist sets of transactions that are schedulable by *strict More-Less*, but not *Half-Half*. In contrast, there also exist sets of transactions that are schedulable by *Half-Half*, but not *strict More-Less*. Compared to *More-Less*, *strict More-Less* can only schedule a strict subset of all the transaction sets that can be scheduled by *More-Less*. This will

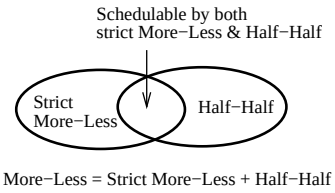

Figure 3. Comparison of *More-Less* and *Half-Half*

be shown later in Section 3.4. Figure 3 illustrates the relationship of *More-Less*, *strict More-Less* and *Half-Half* in terms of schedulable transaction sets. In the figure, the left oval represents sets of transactions schedulable by *strict More-Less*, and the right oval represents sets of transactions schedulable by *Half-Half*. The intersection of the two ovals represents sets of transactions schedulable by both *strict More-Less* and *Half-Half*. The union of the two ovals represents sets of transactions schedulable by *More-Less*.

The crux of the problem then, is to determine an assignment order for a set of transactions such that U_m is minimized. This is left to be discussed later in Sections 3.5, and 3.6. Next, we investigate the impact of jitter on the *Half-Half* and *More-Less* approaches, respectively.

3.2 Jitter Concerns for Update Transactions

With the same jitter bound of δ for all transactions in a set $\mathcal{T} = \{\tau_i\}_{i=1}^m$, it is important to know what the impact of jitter is on deadlines and periods derived from *Half-Half* and *More-Less*. For later notational convenience, we define

$W_i(t) = \sum_{j=1}^{i} C_j \lceil \frac{t}{P_j} \rceil$,

$M_i = max_{\{0<t\leq D_i\}}(t - W_i(t))$.

where $W_i(t)$ denotes the cumulative demands on the processor made by transactions $\tau_1, \tau_2, ..., \tau_i$ over time $[0, t]$ when 0 is a critical instant, and M_i denotes the maximum laxity for transaction τ_i over time $[0, D_i]$.

In the following, we analyze the impact of jitter on *Half-Half*. Impact of jitter on *More-Less* is discussed subsequently.

3.2.1 Jitter Concerns for Half-Half

In the case of Half-Half, assume that a fixed priority scheduling algorithm, e.g., the deadline monotonic scheduling algorithm is used[2]. We have the following theorem.

Theorem 3.2: For a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^m$ with a jitter bound $\delta \geq 0$, and $\alpha_i \leq \alpha_j$ for any $i < j$, the set of transactions can be scheduled by the deadline monotonic scheduling algorithm with deadlines and periods derived from Half-Half if and only if

$$\delta \leq min_{\{1\leq i\leq m\}}(M_i) \tag{2}$$

where $M_i = max_{\{0<t\leq \frac{\alpha_i}{2}\}}(t - \sum_{j=1}^{i} C_j \lceil \frac{t}{\frac{\alpha_j}{2}} \rceil)$ in the case of Half-Half.

Proof: We prove it as follows:

[2]Since the *rate monotonic* [7] scheduling algorithm is a special case of the *deadline monotonic* scheduling algorithm, *deadline monotonic* is assumed here.

1. Suppose that Eq. (2) is true, we need to prove that the set of transactions can be scheduled by the deadline monotonic scheduling algorithm with deadlines and periods derived from Half-Half. Without loss of generality, we focus on transaction τ_i $(1 \leq i \leq m)$. In the case of Half-Half, we know that $P_i = D_i = \frac{\alpha_i}{2}$. Let $0 < t_{max} \leq \frac{\alpha_i}{2}$ such that $t_{max} - W_i(t_{max}) \geq t - W_i(t)$ for $0 < t \leq \frac{\alpha_i}{2}$. Because Eq. (2) holds, we know that $\delta \leq t_{max} - W_i(t_{max}) = t_{max} - \sum_{j=1}^{i} C_j \lceil \frac{t_{max}}{\frac{\alpha_j}{2}} \rceil$. So we have

$$\delta + \sum_{j=1}^{i} C_j \lceil \frac{t_{max}}{\frac{\alpha_j}{2}} \rceil \leq t_{max}. \tag{3}$$

   With a jitter bound δ, the first instances of all transactions arrive at the same time δ in the worst case. Eq. (3) implies that the first instance of τ_i can be completed by time t_{max}. Because $t_{max} \leq \frac{\alpha_i}{2}$, we know that the first instance of τ_i can make its deadline. Due to the fact that the longest response time of τ_i occurs at its first instance, every instance of τ_i can make its deadline if Eq. (2) holds.

   Hence, all transactions in $\mathcal{T}$ are schedulable.

2. Suppose that the set of transactions can be scheduled by the deadline monotonic scheduling algorithm with deadlines and periods derived from Half-Half, we need to prove that Eq. (2) is true.

   Assume that Eq. (2) is not true. There must be transaction τ_i $(1 \leq i \leq m)$ such that $\delta > max_{\{0 < t \leq \frac{\alpha_i}{2}\}}(t - W_i(t))$. That is,

$$\delta > max_{\{0 < t \leq \frac{\alpha_i}{2}\}}(t - \sum_{j=1}^{i} C_j \lceil \frac{t}{\frac{\alpha_j}{2}} \rceil).$$

   Therefore, for all $0 < t \leq \frac{\alpha_i}{2}$, $\delta + \sum_{j=1}^{i} C_j \lceil \frac{t}{\frac{\alpha_j}{2}} \rceil) > t$. This implies that in the worst case when all transactions arrive at the same time δ, τ_i cannot be completed by time $\frac{\alpha_i}{2}$, which is its deadline in the case of Half-Half. It contradicts our assumption that the set of transactions can be scheduled by the deadline monotonic scheduling algorithm with deadlines and periods derived from Half-Half. Hence we have proved that Eq. (2) is true.

Therefore, the set of transactions can be scheduled by the deadline monotonic scheduling algorithm with deadlines and periods derived from Half-Half if and only if Eq. (2) holds. □

The derivation of the maximum jitter bound in Theorem 3.2 requires a maximization of $(t - W_i(t))$ over the continuous variable $t \in [0, \frac{\alpha_i}{2}]$. As discussed in [9], t needs to be checked only a finite number of times. The function $(t - W_i(t))$ is piecewise monotonically increasing except at a finite set of time values called scheduling points. When t is a multiple of one of the periods P_j $(1 \leq j \leq i)$, $(t - W_i(t))$ has a local maximum, which is left continuously increasing and jumps to a lower value to the right. Therefore, we need only test these local maxima to determine if τ_i can make its deadline.

3.2.2 Jitter Concerns for More-Less

Similarly, we have the following theorem for More-Less.

Theorem 3.3: For a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ with a jitter bound $\delta \geq 0$, and deadlines as well as periods derived from More-Less ($D_i \leq D_j$ for any $i < j$), the set of transactions can be scheduled by the deadline monotonic scheduling

algorithm if and only if

$$\delta \leq min_{\{1 \leq i \leq m\}}(M_i) \tag{4}$$

where $M_i = max_{\{0<t\leq D_i\}}(t - \sum_{j=1}^{i} C_j \lceil \frac{t}{P_j} \rceil)$ in the case of More-Less.

Proof: The proof is similar to that of Theorem 3.2. □

3.3 Computation of Deadlines and Periods

We now investigate the problem of computing D_i and P_i with a given transaction order for a set of transactions with known computation times and validity intervals. The following algorithm describes how to compute deadlines and periods of transactions in the presence of jitters.

Input: A jitter bound $\delta \geq 0$, a set of update transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ $(m \geq 1)$ with CPU computation times $\{C_i\}_{i=1}^{m}$ and validity interval lengths $\{\alpha_i\}_{i=1}^{m}$ as well as an assignment order $\tau_1 \rightarrow \tau_2 \rightarrow ... \rightarrow \tau_m$.

Output: Deadlines $\{D_i\}_{i=1}^{m}$ and periods $\{P_i\}_{i=1}^{m}$.

Algorithm 3.1: Determine Deadlines and Periods according to *More-Less*

/* Compute the deadline and period of τ_1 */
$D_1 = \delta + C_1$;
$P_1 = \alpha_1 - D_1$;
/* Compute D_i and P_i for the rest of the tasks in the
descending order of task priorities */

for $i = 2$ **to** m **do**
{
  $R_{i1} = \delta + C_i$; /* Initialize R_{i1}, response time of J_{i1} */
  do { /* Compute R_{i1} iteratively */
    $D_i = R_{i1}$; /* Keep R_{i1} for comparison */
    $R_{i1} = \delta + C_i$; /* Initialize R_{i1} to recompute it */
    /* Next, recompute R_{i1} using D_i */
    for $j = 1$ **to** $i - 1$ **do**
      /* Account for the interference of higher priority tasks */
      { $R_{i1} = R_{i1} + \lceil \frac{D_i}{P_j} \rceil C_j$; }
  } **while** $(R_{i1} \neq D_i)$ **and** $(R_{i1} \leq \frac{\alpha_i}{2})$;
  /* Computation of R_{i1} stops if R_{i1} does not change, or R_{i1} exceeds $\frac{\alpha_i}{2}$ */
  if $(R_{i1} > \frac{\alpha_i}{2})$
  then abort; /* Unschedulable case */
  else $P_i = \alpha_i - D_i$; /* Compute P_i */

i	C_i	α_i	*More-Less*		*Half-Half*
			D_i	P_i	$P_i(D_i)$
1	1	3	1	2	1.5
2	2	20	4	16	10

Table 3. Parameters and results for example 3.1

}

The next example illustrates how Algorithm 3.1 derives deadlines and periods of transactions.

Example 3.1: A set of transactions is given in Table 3 with transaction numbers, computation times, and validity interval lengths. For simplicity, we assume the jitter of each transaction is zero. *Half-Half* and *More-Less* are applied to the transaction set. The resulting deadlines and periods are computed from Algorithm 3.1 and shown in Table 3 with assignment order $\tau_1 \rightarrow \tau_2$, which is the same as the assignment order from the *deadline monotonic* algorithm for the deadlines resulting from *Half-Half*. The CPU workload for *More-Less* is $\frac{1}{2} + \frac{2}{16} = 0.625$, which is less than $\frac{1}{1.5} + \frac{2}{10} = 0.867$, the CPU workload for *Half-Half*.

Example 3.1 shows that *More-Less* can have lower CPU workload than *Half-Half*. Given any set of transactions, does *More-Less* produce better schedulability than *Half-Half* when *More-Less* and *Half-Half* result in different solutions ? This is answered in the affirmative next.

3.4 Comparison of *More-Less* and *Half-Half*

For clarity of presentation, let $W_k^H(t)$ denote the cumulative demands on the processor made by transactions $\tau_1, \tau_2, ..., \tau_k$ over time $[0, t]$ under Half-Half when 0 is a critical instant, i.e.,

$$W_k^H(t) = \sum_{j=1}^{k} \lceil \frac{t}{\frac{\alpha_j}{2}} \rceil C_j. \tag{5}$$

The next theorem states that there are transaction sets that can be scheduled by *Half-Half*, but cannot be scheduled by *strict More-Less*.

Theorem 3.4: Given any set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^m$ $(m \geq 1)$ with $\tau_i \rightarrow \tau_j$ for any $i < j$ and $\delta \geq 0$, if

$$\delta + W_k^H(\frac{\alpha_k}{2}) = \frac{\alpha_k}{2} \qquad (1 \leq k \leq m) \tag{6}$$

and

$$\forall t < \frac{\alpha_k}{2}, \delta + W_k^H(t) > t \qquad (1 \leq k \leq m) \tag{7}$$

hold, then the set of transactions is schedulable by *Half-Half*, but not *strict More-Less*.

Proof: In the worst case, the jitter of each transaction is equal to δ because δ is the jitter bound. Eq. (6) and (7) imply that the first instance of any transaction τ_k $(1 \leq k \leq m)$ cannot complete before time $\frac{\alpha_k}{2}$, but it can complete exactly at time $\frac{\alpha_k}{2}$ under Half-Half in the worst case. We now prove the theorem as follows:

1. Suppose $\delta = 0$. Eq. (6) reduces to

$$W_k^H(\frac{\alpha_k}{2}) = \frac{\alpha_k}{2} \qquad (1 \leq k \leq m). \tag{8}$$

Based on the definition of $W_k^H(t)$ (Eq. (5)), we have

$$\sum_{j=1}^{k} \lceil \frac{\frac{\alpha_k}{2}}{\frac{\alpha_j}{2}} \rceil C_j = \frac{\alpha_k}{2}. \tag{9}$$

That is,

$$\sum_{j=1}^{k} \lceil \frac{\alpha_k}{\alpha_j} \rceil C_j = \frac{\alpha_k}{2} \qquad (1 \leq k \leq m). \tag{10}$$

Similarly, Eq. (7) reduces to

$$\forall t < \frac{\alpha_k}{2}, \sum_{j=1}^{k} \lceil \frac{t}{\frac{\alpha_j}{2}} \rceil C_j > t \qquad (1 \leq k \leq m). \tag{11}$$

We now prove that for all sets of transactions schedulable under *Half-Half*, transaction sets $\mathcal{T}_1 = \{\tau_1\}$ with $C_1 = \frac{\alpha_1}{2}$ are the only transaction sets satisfying Eq. (6) and (7).

First, $\mathcal{T}_1$ satisfies Eq. (10) and (11). It is also schedulable under *Half-Half* because $C_1 = \frac{\alpha_1}{2}$ where $\frac{\alpha_1}{2}$ is the deadline and period for τ_1 under *Half-Half*.

Secondly, we need to prove that $\mathcal{T}_1$ is the only transaction set satisfying Eq. (10) and (11) that is schedulable under *Half-Half*. We prove it by using contradiction. Assume there are other transaction sets $\mathcal{T}_2 = \{\tau_i\}_{i=1}^{m}$ $(m > 1)$ satisfying Eq. (10) and (11). When $k = 1$, we have $C_1 = \frac{\alpha_1}{2}$ from Eq. (10). When $k > 1$, we also have

$$\lceil \frac{\frac{\alpha_k}{2}}{\frac{\alpha_1}{2}} \rceil C_1 + \sum_{j=2}^{k} \lceil \frac{\frac{\alpha_k}{2}}{\frac{\alpha_j}{2}} \rceil C_j = \frac{\alpha_k}{2} \quad (1 < k \leq m) \tag{12}$$

from Eq. (10). We know that $\lceil \frac{\frac{\alpha_k}{2}}{\frac{\alpha_1}{2}} \rceil C_1 \geq \frac{\alpha_k}{2}$ because $C_1 = \frac{\alpha_1}{2}$. Combined with Eq. (12), it implies that $\sum_{j=1}^{k} \lceil \frac{\frac{\alpha_k}{2}}{\frac{\alpha_j}{2}} \rceil C_j > \frac{\alpha_k}{2}$ if $k > 1$, which contradicts Eq. (10). Therefore, $\mathcal{T}_2$ does not exist. This proves that $\mathcal{T}_1$ is the only transaction set satisfying Eq. (6) and (7) that is scedulable under *Half-Half*.

Because $C_1 = \frac{\alpha_1}{2}$, if τ_1 has to complete by its deadline, $D_1 \geq C_1 = \frac{\alpha_1}{2}$ must be true. Because $P_1 = \alpha_1 - D_1$, it implies that $P_1 \leq \frac{\alpha_1}{2}$, i.e., $D_1 \geq P_1$ must hold. Therefore, *strict More-Less* cannot be applied to $\mathcal{T}_1$ because $D_1 < P_1$ cannot be true.

2. Suppose $\delta > 0$. Because of Eq. (6), the set of transactions is schedulable by *Half-Half* because transaction τ_k $(1 \leq k \leq m)$ can complete exactly at time $\frac{\alpha_k}{2}$.

Let us assume that the set of transactions is also schedulable by *strict More-Less*. There must be one or more transactions that have deadlines less than their corresponding periods. Suppose that τ_k $(1 \leq k \leq m)$ is the highest priority transaction with $D_k < P_k$, i.e., $D_k < \frac{\alpha_k}{2}$, which implies that $D_j = P_j = \frac{\alpha_j}{2}$ $(j < k)$. We have

$\delta + \sum_{j=1}^{k-1} \lceil \frac{D_k}{\frac{\alpha_j}{2}} \rceil C_j + C_k \leq D_k$ because τ_k is schedulable. Combined with the definition of $W_k^H(t)$ (Eq. (5)), we have $\delta + W_k^H(D_k) \leq D_k$. That is, $\exists t < \frac{\alpha_k}{2}$, $\delta + W_k^H(t) \leq t$. This contradicts Eq. (7). Therefore, the set of transactions is not schedulable by *strict More-Less*.

Therefore, a set of transactions satisfying Eq. (6) and (7) is schedulable by *Half-Half*, but not *strict More-Less*. □

Based on Theorem 3.4, we define *strict Half-Half* transaction sets as follows:

Definition 3.3: *Strict Half-Half* transaction set: Any set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^m$ $(m \geq 1)$ satisfying Eq. (6) and (7) is a *strict Half-Half* transaction set. In contrast, a *non-strict Half-Half* transaction set is a set of transactions that is schedulable under *Half-Half* but does not satisfy Eq. (7).

From Figure 3, it can be observed that the part of right oval (i.e., the oval for *Half-Half*), which is not common to both ovals, represents *strict Half-Half*. The intersection of the two ovals represents *non-strict Half-Half*. In terms of schedulable transaction sets, *strict Half-Half* = *Half-Half* − *strict More-Less*. It should be noted that *Half-Half* is only a special case of *More-Less*. So a set of transactions satisfying Eq. (6) and (7) can also be scheduled by *More-Less*. However, for *non-strict Half-Half* transaction sets that are schedulable by Half-Half, the next theorem implies that *strict More-Less* always outperforms *Half-Half* in terms of minimizing CPU workload.

Theorem 3.5: If any *non-strict Half-Half* transaction set $\mathcal{T} = \{\tau_i\}_{i=1}^m$ $(m \geq 1)$ with a jitter bound $\delta \geq 0$ can be scheduled to guarantee data freshness using any fixed priority scheduling algorithm based on deadlines and periods derived from *Half-Half*, then the same set of transactions can also be scheduled by the *deadline monotonic* algorithm based on deadlines and periods derived from *strict More-Less*.

Proof: If $m = 1$, it is trivial. Let us look at the case of $m > 1$.

Without loss of generality, assume that transaction priorities are assigned in the order of $\tau_1 \rightarrow \tau_2 \rightarrow ... \rightarrow \tau_m$ by *Half-Half*. Let us assume that the same priority order is retained by *strict More-Less*. Let D_i^H and P_i^H denote the deadline and period of transaction τ_i in *Half-Half*, and D_i^M and P_i^M denote the deadline and period of transaction τ_i in *strict More-Less*, respectively. We know that $D_i^H = P_i^H = \frac{\alpha_i}{2}$. Since the set of transactions can be scheduled by a fixed priority scheduling algorithm based on *Half-Half*, we will prove that $D_i^H \geq D_i^M$ and $P_i^H \leq P_i^M$ for $1 \leq i \leq m$, and there is at least one transaction τ_k $(1 \leq k \leq m)$ with $D_k^M < P_k^M$.

Assume that τ_k $(1 \leq k \leq m)$ is the highest priority transaction that does not satisfy Eq. (7). For $1 \leq j \leq k-1$, let $D_j^M = D_j^H$ and $P_j^M = P_j^H$. It is clear that $\exists (t < \frac{\alpha_k}{2})$ such that $\delta + C_k + \sum_{j=1}^{k-1} (\lceil \frac{t}{P_j^M} \rceil C_j) \leq t$, i.e., $\delta + \sum_{j=1}^{k} (\lceil \frac{t}{\frac{\alpha_j}{2}} \rceil C_j) \leq t$. Therefore, we have $\delta + W_k^H(t) \leq t$.

Let $D_k^M = t$. This implies that $D_k^M < \frac{\alpha_k}{2}$ and $P_k^M > \frac{\alpha_k}{2}$, i.e., $D_k^M < P_k^M$. For all $k < j \leq m$, let $D_j^M = D_j^H$ and $P_j^M = P_j^H$. Now we have deadlines and periods derived from *strict More-Less* for all transactions. It is obvious that the first instance of τ_i $(1 \leq i \leq m)$ can make its deadline based on the *deadline monotonic* scheduling algorithm. It directly follows from Lemma 3.2 that the set of transactions with deadlines and periods derived from *strict More-Less* can be scheduled by the *deadline monotonic* in the presence of a jitter bound. □

i	C_i	α_i	*Strict More-Less*		*Half-Half*
			D_i	P_i	$P_i(D_i)$
1	1	4	1	3	2
2	1	5	2	3	2.5
3	1	8	3	5	4
4	1	20	9	11	10

Table 4. Parameters and results of Example 3.2

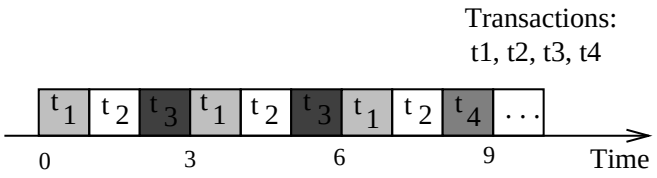

Figure 4. A solution produced by *More-Less*

From Theorem 3.5, if there is a feasible solution based on *Half-Half* for a set of transactions and Eq. (7) does not hold, there must be a feasible solution based on *strict More-Less* with lower CPU workload. However, the *converse* is not true. This is illustrated by Example 3.2.

Example 3.2: Transactions are listed in Table 4 with transaction numbers, computation times, validity interval lengths. *Half-Half* and *strict More-Less* are applied to the transaction set, and resulting deadlines and periods are shown in Table 4. For simplicity, we assume that jitter of a transaction is zero. It is clear from Table 4 that the transaction set resulting from *Half-Half* is non-schedulable because its CPU workload is $\frac{1}{2} + \frac{1}{2.5} + \frac{1}{4} + \frac{1}{10} = 1.25 > 1.0$. However, transactions with periods resulting from *strict More-Less* is schedulable by assigning priorities $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \tau_4$. In this case, the resulting CPU workload is $\frac{1}{3} + \frac{1}{3} + \frac{1}{5} + \frac{1}{11} = 0.957$. Figure 4 shows that the first instance of every transaction in the set can meet its deadline, which indicates that the transaction set is schedulable according to Lemma 3.2. However, an assignment order $\tau_2 \rightarrow \tau_1 \rightarrow \tau_3 \rightarrow \tau_4$ under *More-Less* would not be able to produce a feasible solution according to Algorithm 3.1. Specifically, the algorithm produces periods 4, 2, 4 for transactions τ_2, τ_1, and τ_3, respectively, and those three transactions have higher priorities than τ_4. With such periods and transaction computation times, it is easy for readers to verify that transactions τ_2, τ_1, and τ_3 would consume 100% of the CPU time, and there is no CPU capacity left for τ_4. The algorithm would fail to produce a feasible deadline for τ_4 because R_{41} exceeds $\frac{\alpha_4}{2}$. This indicates that assignment orders of transactions can significantly affect the schedulability of transactions. □

It is clear from Theorem 3.5 that any conditions sufficient to guarantee the schedulability of a set of transactions using *Half-Half* while Eq. (7) does not hold must be sufficient to guarantee the schedulability using *strict More-Less*. The following lemma gives a sufficient condition for the schedulability of transactions based on *strict More-Less*.

Lemma 3.3: Given any set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ $(m \geq 1)$ with $\tau_i \rightarrow \tau_j$ for any $i < j$, let $W_i^H(t) = \sum_{j=1}^{i} \lceil \frac{t}{\frac{\alpha_j}{2}} \rceil C_j$ under Half-Half. If

$$\exists t \leq \frac{\alpha_i}{2}, \quad \delta + W_i^H(t) \leq t \qquad (1 \leq i \leq m) \tag{13}$$

and

$$\exists 1 \leq k \leq m, \exists t < \frac{\alpha_k}{2}, \quad \delta + W_k^H(t) \leq t \tag{14}$$

hold, then the set of transactions is schedulable by the deadline monotonic scheduling algorithm under *strict More-Less*.

Proof: It should be noted that if Eq. (13) holds, the first instance of transaction τ_i $(1 \leq i \leq m)$ can complete at time $t \leq \frac{\alpha_i}{2}$ under Half-Half. Then $\mathcal{T}$ is schedulable by a fixed priority scheduling algorithm with deadlines and periods derived from Half-Half with priority order $\tau_i \rightarrow \tau_j$ for any $i < j$. If Eq. (14) holds, there is at least one transaction τ_k whose first instance can complete at time $t < \frac{\alpha_k}{2}$. From Theorem 3.5, we know that the transaction set $\mathcal{T}$ is schedulable by the deadline monotonic scheduling algorithm under *strict More-Less*. □

Lemma 3.3 is interesting because it gives the sufficient condition for the subset of Half-Half that is also schedulable under strict More-Less. It should be noted that conditions in Lemma 3.3 are only sufficient conditions for scheduling a set of transactions based on *strict More-Less*; they are not the necessary conditions. However, Eq. (13) is both a necessary and sufficient condition for scheduling a set of transactions with fixed priority scheduling algorithms based on *Half-Half*, that is, if Eq. (13) does not hold, a set of transactions is not schedulable based on *Half-Half*. However, it may still be schedulable using *More-Less*. As illustrated in Example 3.2, assignment orders in *More-Less* may have a significant impact on the schedulability of transactions. How to choose an appropriate assignment order to determine deadlines and periods remains a problem. An optimal assignment order is desirable for *More-Less* to guarantee schedulability and minimize the CPU workload of transactions.

3.5 *More-Less* Approach: An Optimal Approach in a Restricted Case

As far as we know, there is no known solution to solve the previous non-linear programming problem corresponding to producing optimal periods and deadlines under *More-Less*. The complexity arises from not only the non-linearity, but also the permutation of m transactions (i.e., the assignment order of the m transactions), which is $O(m!)$. If we enumerate all the permutations of m transactions to find the one with minimized CPU workload, all $m!$ solutions would have to be examined. It is obviously not efficient when the transaction set is large.

We now begin to examine the issue of finding an optimal assignment order for *More-Less*. For simplicity of presentation, we assume that jitter of all transactions are constrained to be 0 (i.e., $\delta = 0$). However, our results can also be extended to the situation in which $\delta > 0$. We first consider the problem with the following constraint:

Restriction (1): $\sum_{i=1}^{m} C_i \leq min(\frac{\alpha_j}{2})$ $(1 \leq j \leq m)$.

Under this restriction, the first instance of all transactions can complete before half of the shortest validity interval. Given any assignment order of transactions, this implies that no higher priority transactions can recur before the first instance of the lowest priority transaction completes. Otherwise, suppose J_{i2} (the 2nd instance of transaction τ_i) $(1 \leq i \leq m)$ is the first

recurring instance, and it occurs at time t before the first instance of the lowest priority transaction completes. It implies that $t \geq P_i$. Because $P_i \geq \frac{\alpha_i}{2}$ according to *More-Less*, we have $t \geq \frac{\alpha_i}{2}$. Because not all the first instances from all transactions have completed yet, $t < \sum_{i=1}^{m} C_i$. Therefore we can conclude that $\frac{\alpha_i}{2} < \sum_{i=1}^{m} C_i$, which contradicts Restriction (1). Hence $n_{ij} = 1\ (1 \leq i \leq m\ \&\ 1 \leq j \leq i)$, i.e., no higher priority transactions can recur before the first instance of the lowest priority transaction completes. Due to the short execution time of sensor transactions and relatively long validity interval length in many real applications (e.g., avionics system in [6], air traffic control, aircraft mission processor, and spacecraft control in [8]), Restriction (1) is reasonable in many cases. In the rest of Section 3.5, we assume that Restriction (1) holds. In the rest of the paper, we also assume that transactions are ordered so that $\tau_i \rightarrow \tau_j$ for $i < j$ unless specified otherwise.

3.5.1 *More-Less* Approach: Optimal Assignment Order for Two Transactions

To motivate our approach to determining the ordering of transactions, we first study the characteristics of a set of two transactions: τ_1 and τ_2. The question we are trying to answer is, which one should precede the other? Two cases are examined:

1. $\tau_1\tau_2$: $\tau_1 \rightarrow \tau_2$

$$\begin{cases} P_1 = \alpha_1 - C_1 \\ P_2 = \alpha_2 - (C_1 + C_2) \end{cases} \tag{15}$$

2. $\tau_2\tau_1$: $\tau_2 \rightarrow \tau_1$

$$\begin{cases} P_2 = \alpha_2 - C_2 \\ P_1 = \alpha_1 - (C_1 + C_2) \end{cases} \tag{16}$$

In the above two cases, it should be noted that the higher priority transaction only occurs once before the first instance of the lower priority transaction completes because Restriction (1) holds. Let U_{12} and U_{21} denote the CPU workload of transactions τ_1 and τ_2 in cases $\tau_1\tau_2$ and $\tau_2\tau_1$, respectively. Now we have

$$\begin{cases} U_{12} = \sum_{i=1}^{2} \frac{C_i}{P_i} = \frac{C_1}{\alpha_1 - C_1} + \frac{C_2}{\alpha_2 - (C_1 + C_2)} \\ U_{21} = \sum_{i=1}^{2} \frac{C_i}{P_i} = \frac{C_2}{\alpha_2 - C_2} + \frac{C_1}{\alpha_1 - (C_1 + C_2)} \end{cases} \tag{17}$$

Without loss of generality, assume we want to show that $U_{12} \leq U_{21}$, i.e.,

$$\frac{C_1}{\alpha_1 - C_1} + \frac{C_2}{\alpha_2 - (C_1 + C_2)} \leq \frac{C_2}{\alpha_2 - C_2} + \frac{C_1}{\alpha_1 - (C_1 + C_2)}. \tag{18}$$

We now study the conditions that satisfy Eq. (18).

Let L_i denote the validity interval slack of transaction τ_i, i.e., $L_i = \alpha_i - C_i$. Also let $\Delta\alpha_{ij} = \alpha_i - \alpha_j$, $\Delta L_{ij} = L_i - L_j$, and $\Delta C_{ij} = C_i - C_j$. It is obvious that $\Delta\alpha_{ji} = -\Delta\alpha_{ij}$, $\Delta L_{ji} = -\Delta L_{ij}$, and $\Delta C_{ji} = -\Delta C_{ij}$. Suppose an assignment order $\tau_i \rightarrow \tau_j$ of two adjacent transactions τ_i and τ_j is given, let U_{ij} denote CPU workload of τ_i and τ_j with $\tau_i \rightarrow \tau_j$, i.e., $U_{ij} = \frac{C_i}{P_i} + \frac{C_j}{P_j}$. We now introduce the following theorem.

Theorem 3.6: Given any set of transactions $\mathcal{T} = \{\tau_k\}_{k=1}^{m}$ $(m = 2)$, for two transactions τ_i and τ_j $(1 \leq i, j \leq m \quad and \quad i \neq j)$, if

1.Restriction (1) holds (i.e., $\frac{\alpha_i}{2} \geq C_i + C_j$ and $\frac{\alpha_j}{2} \geq C_i + C_j$).

2.$\Delta\alpha_{ji} \geq 0$ and $\Delta C_{ji} \leq 2\Delta\alpha_{ji}$, i.e., for $\alpha_j \geq \alpha_i$, the increase of computation time is at most twice the increase in validity interval length,

then $U_{ij} \leq U_{ji}$, i.e.,
$\frac{C_i}{\alpha_i - C_i} + \frac{C_j}{\alpha_j - (C_i + C_j)} \leq \frac{C_j}{\alpha_j - C_j} + \frac{C_i}{\alpha_i - (C_i + C_j)}$.

Theorem 3.6, proved in Appendix, is generalized in Section 3.5.2 to provide restrictions under which optimal solutions can be easily produced. It has the following properties under Restriction (1): *stability* and *transitivity* as described below.

Definition 3.4: Stability: No matter how many transactions are assigned higher priority than two adjacent transactions τ_i and τ_j (i.e., no other transactions exist with priority between τ_i and τ_j), the ordering of τ_i and τ_j is *stable* which means U_{ij} is at most equivalent to U_{ji}.

For a set of transactions $\mathcal{T} = \{\tau_k\}_{k=1}^{m}$ $(m > 2)$, it should be noted that $U_{ij} \leq U_{ji}$ may be affected by higher priority transactions. This is because higher priority transactions have impact on the derived periods of τ_i and τ_j. For example, if P_i and P_j are changed because of changes of transaction priorities, $U_{ij} \leq U_{ji}$ may not always hold. But in general, the following property holds.

Property 1. If transactions τ_i and τ_j satisfy the two conditions in Theorem 3.6, then the ordering of τ_i and τ_j is *stable*, i.e., $U_{ij} \leq U_{ji}$ always holds.

Proof: To prove that $U_{ij} \leq U_{ji}$ always holds, we need to prove that no matter how many transactions have been assigned higher priorities than τ_i and τ_j, $U_{ij} \leq U_{ji}$ always holds.

Suppose k transactions, $\tau_1, \tau_2, ..., \tau_k$, have been assigned higher priorities than τ_i and τ_j. The sum of their computation times is $\sum_{l=1}^{k} C_l = C$ $(C \geq 0)$. Assume that $U_{ij} \leq U_{ji}$ holds when $C = 0$, i.e., there are no transactions with priority higher than τ_i and τ_j. We want to prove that $U_{ij} \leq U_{ji}$ also holds when $C > 0$, i.e., $\frac{C_i}{\alpha_i - C - C_i} + \frac{C_j}{\alpha_j - (C + C_i + C_j)} \leq \frac{C_j}{\alpha_j - C - C_j} + \frac{C_i}{\alpha_i - (C + C_i + C_j)}$. Consider a new transaction system with τ_i^* and τ_j^* Let $\alpha_i^* = \alpha_i - C$ and $\alpha_j^* = \alpha_j - C$, but $C_i^* = C_i$ and $C_j^* = C_j$ thus

$$\Delta\alpha_{ji}^* = \alpha_j^* - \alpha_i^* = \alpha_j - \alpha_i = \Delta\alpha_{ji}. \tag{19}$$

We know that $\Delta C_{ji} \leq 2\Delta\alpha_{ji}$ for $C = 0$ because of Condition 2 in Theorem 3.6. Combined with Eq. (19), $\Delta C_{ji} \leq 2\Delta\alpha_{ji} \Leftrightarrow \Delta C_{ji} \leq 2\Delta\alpha_{ji}^*$. Replacing $\Delta\alpha_{ji}$, α_i and α_j with $\Delta\alpha_{ji}^*$, α_i^* and α_j^*, respectively, in the conditions of Theorem 3.6, we have $\frac{C_i}{\alpha_i^* - C_i} + \frac{C_j}{\alpha_j^* - (C_i + C_j)} \leq \frac{C_j}{\alpha_j^* - C_j} + \frac{C_i}{\alpha_i^* - (C_i + C_j)}$. That is, $U_{ij} \leq U_{ji}$ also holds when $C > 0$. □

Definition 3.5: Transitivity: If τ_i preceding τ_j results in $U_{ij} \leq U_{ji}$, and τ_j preceding τ_k results in $U_{jk} \leq U_{kj}$, then τ_i preceding τ_k results in $U_{ik} \leq U_{ki}$.

Property 2. Transactions satisfying conditions in Theorem 3.6 maintain *transitivity*.

Proof: Given transactions τ_i, τ_j and τ_k, suppose $\Delta C_{ji} \leq 2\Delta\alpha_{ji}$ and $\Delta C_{kj} \leq 2\Delta\alpha_{kj}$. Because $2\Delta\alpha_{ki} = 2(\alpha_k - \alpha_i) = 2(\alpha_k - \alpha_j) + 2(\alpha_j - \alpha_i) = 2\Delta\alpha_{kj} + 2\Delta\alpha_{ji} \geq \Delta C_{kj} + \Delta C_{ji} = \Delta C_{ki}$, we have $U_{ik} \leq U_{ki}$. □

Determining the conditions necessary from Theorem 3.6 for $U_{ij} \leq U_{ji}$ is computationally efficient because $\Delta\alpha_{ji}$ and ΔC_{ij} are computable in polynomial time.

Discussion

In Theorem 3.6, $\Delta\alpha_{ji} \geq 0$ and $\Delta C_{ji} \leq 2\Delta\alpha_{ji}$ include two cases:

1. $\Delta\alpha_{ji} \geq 0$ and $\Delta\alpha_{ji} \geq \Delta C_{ji}$, which implies $\alpha_i \leq \alpha_j$ and $\alpha_i - C_i \leq \alpha_j - C_j$, i.e., α_i and $\alpha_i - C_i$ order transaction priorities in the *same* way.

2. $\Delta\alpha_{ji} \geq 0$ and $2\Delta\alpha_{ji} \geq \Delta C_{ji} \geq \Delta\alpha_{ji}$, which implies $\alpha_i \leq \alpha_j$ and $\alpha_j - C_j \leq \alpha_i - C_i$, i.e., α_i and $\alpha_i - C_i$ do *not* order transaction priorities in the same way.

τ_i preceding τ_j produces lower CPU workload in the above two cases. Thus, $\alpha_i - C_i$ values of transactions may not produce the best assignment order. Said differently, the Least Slack First assignment algorithm may not produce the lowest workload.

3.5.2 *More-Less* Approach: Optimal Assignment Order for m Transactions

To generalize the comparison of two transactions, we need to examine a set of transactions $\{\tau_i\}_{i=1}^{m}$ with $m > 2$. We first introduce the second restriction in this paper.

Restriction (2):

$$\begin{cases} \alpha_1 \leq \alpha_2 \leq ... \leq \alpha_m \\ \Delta C_{i+1,i} \leq 2\Delta\alpha_{i+1,i} \ (i = 1, 2, .., m-1) \end{cases}$$

The next theorem proposes an optimal solution under Restrictions (1) and (2).

Theorem 3.7: Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$, if Restrictions (1) and (2) hold then an assignment order named *Shortest Validity First(SVF)*, which assigns orders to transactions in the *inverse* order of validity interval length and resolves ties in favor of a transaction with less *slack*[3], results in the optimal CPU workload among all possible assignment orders of the *More-Less* approach.

Proof: It should be noted that there may be more than one transaction order resulting from SVF if some transactions have the same validity interval length and computation time. But the transaction orders resulting from SVF have the same CPU workload.

We need to prove that the transaction ordering scheme from SVF results in the minimal CPU workload. From Restriction (2) and Theorem 3.6, we know that $U_{i,i+1} \leq U_{i+1,i}$ $(1 \leq i \leq m-1)$, and this is stable and transitive. Suppose there is an optimal assignment ordering K_{opt} resulting from an order different from SVF. But order K_{opt} can always be achieved by a sequence of swappings of priorities of two adjacent transactions in our SVF scheme. From the *stability* and *transitivity* of Theorem 3.6, we know that every swap of orders of two adjacent transactions in the SVF scheme would result in higher CPU workload if these two transactions do not have the same validity interval length and computation time. For all the swaps of

[3]As in Table 1, slack L_i for transaction τ_i is defined as $\alpha_i - C_i$.

i	C_i	α_i	*More-Less*		*Half-Half*
			D_i	P_i	$P_i(D_i)$
1	1	8	1	7	4
2	1	10	2	8	5
3	1	12	3	9	6

Table 5. Illustration of an optimal solution

U_{123}	U_{132}	U_{213}	U_{231}	U_{312}	U_{321}
0.379	0.386	0.389	0.411	0.400	0.416

Table 6. CPU workload of all possible orderings

orders of two adjacent transactions in the SVF scheme, there must be at least one swap of two transactions that do not have the same validity interval length and computation time. Otherwise, order K_{opt} is only one of the SVF orders, which contradicts the assumption that K_{opt} results from an order different from SVF. Thus order K_{opt} has higher CPU workload than the SVF scheme. This contradicts the assumption that K_{opt} is optimal. Therefore we have proved that transaction ordering scheme based on SVF results in the optimal CPU workload. □

Example 3.3: In Table 5, a set of transactions is shown that satisfies Restrictions (1) and (2), therefore an assignment order $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$ results in an optimal solution for *More-Less*. *Half-Half* and *More-Less* are applied to the transaction set, respectively, and the resulting deadlines and periods are shown in Table 5. The resulting CPU workload of the solution from *More-Less* is $\frac{1}{7} + \frac{1}{8} + \frac{1}{9} = 0.379$. This is an optimal CPU workload among all the priority assignments of *More-Less*, and it is much lower than the CPU workload of the solution from *Half-Half*, which is $\frac{1}{4} + \frac{1}{5} + \frac{1}{6} = 0.62$. CPU workloads of all possible assignment orders are listed in Table 6 in which U_{XYZ} represents workload of assignment order $\tau_X \rightarrow \tau_Y \rightarrow \tau_Z$. We can see that SVF does result in the optimal CPU workload in this case. □

The next example illustrates that SVF does not produce an optimal solution if Restriction (2) does not hold.

Example 3.4: In Table 7, it is obvious that the set of transactions does not satisfy Restriction (2) because $\frac{\Delta C_{21}}{\Delta \alpha_{21}} = \frac{4-1}{11-10} = 3 > 2$, although Restriction (1) holds. In this case, an assignment order according to SVF ($\tau_1 \rightarrow \tau_2$) does not result in an optimal solution for *More-Less*. Resulting deadlines and periods from different assignment orders under *More-Less* are shown in Table 7. In this case, the resulting CPU workload of SVF is $\frac{1}{9} + \frac{4}{6} = 0.778$, and the other order results in a CPU workload of $\frac{1}{5} + \frac{4}{7} = 0.771$. □

So, clearly, when Restriction (1) holds but Restriction (2) does not hold, SVF may not be an optimal solution. But it is interesting to note that SVF produces a CPU workload which is close to the optimal in such situations. This is the issue that is examined next.

i	C_i	α_i	$\tau_1 \rightarrow \tau_2$		$\tau_2 \rightarrow \tau_1$	
			D_i	P_i	D_i	P_i
1	1	10	1	9	5	5
2	4	11	5	6	4	7

Table 7. SVF is non-optimal case

3.6 *More-Less* Approach: An Approximate Solution and Its Bound

In this subsection, we explore the implication of using SVF even when Restriction (2) in Theorem 3.7 does not hold, but Restriction (1) holds. We will show that SVF can provide a CPU workload bounded within a certain range of that of the optimal solution. This is analyzed through the help of transaction *partitioning*, a powerful technique which can help derive the CPU workload bound when using SVF as an approximation of the optimal assignment order.

Definition 3.6: Partition: Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ $(m \geq 1)$, if a transaction τ_k $(1 \leq k \leq m)$ is partitioned into n $(n > 1)$ independent subtransactions $\{\tau_{kj}\}_{j=1}^{n}$ with computation time $\{C_{kj}\}_{j=1}^{n}$ $(\sum_1^n C_{kj} = C_k)$, and validity interval length $\{\alpha_{kj}\}_{j=1}^{n}$ $(\alpha_{kj} = \alpha_k)$, then the set of transactions $\{\tau_{kj}\}_{j=1}^{n}$ is a *partition* of τ_k, and the resulting set of transactions $\{\tau_i\}_{i=1}^{k-1} \cup \{\tau_{kj}\}_{j=1}^{n} \cup \{\tau_i\}_{i=k+1}^{m}$ is a *partition-transformed* set of the original transaction set.

It should be noted that *partition-transformation* is *transitive*. For example, if transaction set $\mathcal{T}_\mathcal{B}$ is a partition-transformed set of transaction set $\mathcal{T}_\mathcal{A}$, and transaction set $\mathcal{T}_\mathcal{C}$ is a partition-transformed set of transaction set $\mathcal{T}_\mathcal{B}$, then transaction set $\mathcal{T}_\mathcal{C}$ is a partition-transformed set of transaction set $\mathcal{T}_\mathcal{A}$.

We now investigate the impact of *partitioning* on minimizing the CPU workload of a transaction set. We want to understand whether partitioning transactions into smaller subtransactions with shorter computation times would produce optimal solutions with lower CPU workload. The following theorem holds even when Restriction (1) is not satisfied.

Theorem 3.8: Given any set of transactions $\mathcal{T}_\mathcal{O} = \{\tau_i\}_{i=1}^{m}$, assume that a transaction τ_k $(1 \leq k \leq m)$ can be partitioned into n $(n \geq 2)$independent subtransactions $\{\tau_{kj}\}_{j=1}^{n}$ with $C_k = \sum_{j=1}^{n} C_{kj}$ and $\alpha_{kj} = \alpha_k$ $(1 \leq j \leq n)$. Let the partition-transformed transaction set be $\mathcal{T}_\mathcal{P}$. Then for any solution generated by *More-Less*, the optimal CPU workload of $\mathcal{T}_\mathcal{P}$ is less than the optimal CPU workload of $\mathcal{T}_\mathcal{O}$.

Proof: For an optimal solution ${\mathcal{S}_\mathcal{O}}^{opt}$ of $\mathcal{T}_\mathcal{O}$ generated by *More-Less* with assignment order $\tau_1 \rightarrow \tau_2 \rightarrow ... \rightarrow \tau_m$, if a transaction $\tau_k \in \mathcal{T}_\mathcal{O}$ $(1 \leq k \leq m)$ can be partitioned into n subtransactions $\tau_{k1}, ..., \tau_{kn}$, $\mathcal{T}_\mathcal{O}$ is transformed into a transaction set $\mathcal{T}_\mathcal{P} = \{\tau_1^{'}, .., \tau_{k-1}^{'}, \tau_{k1}^{'}, ..., \tau_{kn}^{'}, \tau_{k+1}^{'}, ..., \tau_m^{'}\}$ where $\tau_j^{'} = \tau_j$ $(j \neq k)$ and $\tau_{ki}^{'} = \tau_{ki}$ $(1 \leq i \leq n)$. Based on *More-Less*,

we can obtain a feasible solution $\mathcal{S}_{\mathcal{P}}$ from ${\mathcal{S}_{\mathcal{O}}}^{opt}$ immediately with

$$\begin{cases} D_j^{'} = D_j(j < k) \\ D_{ji}^{'} < D_j(j = k \ \& \ 1 \leq i \leq n-1) \\ D_{ji}^{'} = D_j(j = k \ \& \ i = n) \\ D_j^{'} = D_j(j > k) \end{cases} \tag{20}$$

by assigning priorities in the order of $\tau_1^{'} \rightarrow ... \rightarrow \tau_{k-1}^{'} \rightarrow \tau_{k1}^{'} \rightarrow ... \rightarrow \tau_{kn}^{'} \rightarrow \tau_{k+1}^{'} \rightarrow ... \rightarrow \tau_m^{'}$. Thus, we know that

$$\begin{cases} P_j^{'} = P_j(j < k) \\ P_{ji}^{'} > P_j(j = k \ \& \ 1 \leq i \leq n-1) \\ P_{ji}^{'} = P_j(j = k \ \& \ i = n) \\ P_j^{'} = P_j(j > k) \end{cases} \tag{21}$$

We know that $\mathcal{T}_{\mathcal{P}}$ with the above $\{D_i^{'}\}$ and $\{P_i^{'}\}$ can be scheduled because deadlines and periods are produced from a feasible solution, ${\mathcal{S}_{\mathcal{O}}}^{opt}$. Considering that

$$U_{\mathcal{T}_{\mathcal{O}}}^{opt} = \sum_{i=1}^{m} \frac{C_i}{P_i}, \tag{22}$$

and

$$U_{\mathcal{T}_{\mathcal{P}}} = \sum_{i=1}^{k-1} \frac{C_i}{P_i} + \sum_{i=1}^{n} \frac{C_{ki}^{'}}{P_{ki}^{'}} + \sum_{i=k+1}^{m} \frac{C_i}{P_i}, \tag{23}$$

we know that $U_{\mathcal{T}_{\mathcal{O}}}^{opt} > U_{\mathcal{T}_{\mathcal{P}}}$. Because $U_{\mathcal{T}_{\mathcal{P}}}^{opt}$, the optimal CPU workload of $\mathcal{T}_{\mathcal{P}}$, is less than or equal to $U_{\mathcal{T}_{\mathcal{P}}}$, we can conclude that $U_{\mathcal{T}_{\mathcal{O}}}^{opt} > U_{\mathcal{T}_{\mathcal{P}}}^{opt}$. This proves the theorem. □

Theorem 3.8 is important because it says that a partition-transformed set has lower optimal CPU workload than the optimal CPU workload of its original transaction set. Theorem 3.8 can be applied repeatedly to every transaction in $\mathcal{T}_{\mathcal{O}}$. This generates a "finer" transaction set with even lower optimal CPU workload. We will show later in this paper how *partitioning* can be used to analyze *More-Less*.

Given a set of transactions $\mathcal{T}$ which satisfies Restriction (1) but does not satisfy Restriction (2), we can partition transactions which violate Restriction (2) into a set of subtransactions such that the partition-transformed transaction set $\mathcal{T}_{\mathcal{P}}$ satisfies Restriction (2). The optimal CPU workload of the partition-transformed transaction set ($U_{\mathcal{T}_{\mathcal{P}}}^{opt}$) can be obtained from Theorem 3.7, and this is less than the optimal CPU workload of the original transaction set ($U_{\mathcal{T}}^{opt}$) as per Theorem 3.8. Thus, for any given solution $\mathcal{S}$ of $\mathcal{T}$ and its CPU workload $U_{\mathcal{S}}$, $U_{\mathcal{S}} - U_{\mathcal{T}}^{opt} \leq U_{\mathcal{S}} - U_{\mathcal{T}_{\mathcal{P}}}^{opt}$ because $U_{\mathcal{T}}^{opt} \geq U_{\mathcal{T}_{\mathcal{P}}}^{opt}$.

Definition 3.7: Partition/Merge: Given any set of transactions $\mathcal{T} = \{\tau_1, \tau_2, .., \tau_m\}$ with $\alpha_1 \leq \alpha_2 \leq ... \leq \alpha_m$, if Restriction (1) holds but Restriction (2) does not hold for $\mathcal{T}$, we can reconstruct the transaction set by partitioning the computation time of transactions so that Restriction (2) holds.

1. *Partitioning of one transaction:* If there is one transaction τ_k with $\alpha_{k-1} \leq \alpha_k$ and $C_k > C_{k-1} + 2(\alpha_k - \alpha_{k-1})$, in which case Restriction (2) does not hold, we can partition the computation time C_k into n ($n \geq 2$) parts that satisfy $\frac{C_k}{n} \leq C_{k-1} + 2(\alpha_k - \alpha_{k-1})$ (which again implies $\Delta C_{k,k-1} \leq 2\Delta\alpha_{k,k-1}$). We can consider τ_k to consist of a set of n subtransactions: $\mathcal{T}_{\tau_k} = \{\tau_{k1}, \tau_{k2}, .., \tau_{kn}\}$, in which $\alpha_{ki} = \alpha_k$ and $C_{ki} = \frac{C_k}{n}$ ($1 \leq i \leq n$). We

denote $\mathcal{T}_{\tau_k} = \mathcal{P}(\tau_k)$. Let us substitute the set of transactions $\tau_{k1}, \tau_{k2}, .., \tau_{kn}$ for transaction τ_k and form a new set of transactions $\mathcal{T}_k = \{\tau_1, .., \tau_{k-1}, \tau_{k1}, \tau_{k2}, .., \tau_{kn}, \tau_{k+1}, .., \tau_m\}$. If we assign orders of transactions in $\mathcal{T}_k$ according to SVF and derive periods based on Eq. (1), it is easy to see that $D_{ki} \leq D_k$, that is, $P_{ki} \geq P_k$.

2. *Partitioning of more than one transaction:* If there are multiple adjacent transactions that do not satisfy Restriction (2), they are partitioned in the same way and the set of old transactions is transformed into a set of new transactions $\mathcal{T}_\mathcal{P} = \{\tau_1, \tau_2, .., \tau_{m_p}\}$ $(m_p \geq m)$. Transactions in $\mathcal{T}_\mathcal{P}$ now satisfy Restriction (2), thus the optimal solution of transaction set $\mathcal{T}_\mathcal{P}$ can be achieved by applying theorem 3.7.

Merge (denoted as $\mathcal{P}^{-1}$) is the inverse function of *Partition*. If $\mathcal{T}_k = \mathcal{P}(\tau_k)$, then $\tau_k = \mathcal{P}^{-1}(\mathcal{T}_k)$.

Let $U_\mathcal{T}^{opt}$ and $U_{\mathcal{T}_k}^{opt}$ denote the optimal solution of $\mathcal{T}$ and $\mathcal{T}_k$, respectively. It is obvious that

$$U_{\mathcal{T}_k}^{opt} = \sum_{i=1}^{k-1} \frac{C_i}{P_i} + \sum_{i=k+1}^{m} \frac{C_i}{P_i} + \sum_{j=1}^{n} \frac{\frac{C_k}{n}}{P_{kj}}. \tag{24}$$

As per Theorem 3.8, $U_{\mathcal{T}_k}^{opt} \leq U_\mathcal{T}^{opt}$. Applying Theorem 3.8 repeatedly to $\mathcal{T}$, we know that the CPU workload of the optimal solution of $\mathcal{T}_\mathcal{P}$, $U_{\mathcal{T}_\mathcal{P}}^{opt}$, satisfies

$$U_{\mathcal{T}_\mathcal{P}}^{opt} \leq U_\mathcal{T}^{opt}. \tag{25}$$

Theorem 3.9: Given a set of transactions $\mathcal{T}$ which satisfies Restriction (1), let $U_\mathcal{T}^{opt}$, $U_{\mathcal{T}_\mathcal{P}}^{opt}$, and $U_{\mathcal{S}^*}$ denote the CPU workload of an optimal solution of $\mathcal{T}$, the optimal solution of $\mathcal{T}_\mathcal{P}$, and the approximate solution $\mathcal{S}^*$ of $\mathcal{T}$ derived from *Shortest Validity First (SVF)*, respectively. The following inequality holds:

$$U_{\mathcal{S}^*} \geq U_\mathcal{T}^{opt} \geq U_{\mathcal{T}_\mathcal{P}}^{opt}. \tag{26}$$

Proof: $U_{\mathcal{S}^*} \geq U_\mathcal{T}^{opt}$ because $U_\mathcal{T}^{opt}$ is the optimal CPU workload of the same set of transactions. We know $U_\mathcal{T}^{opt} \geq U_{\mathcal{T}_\mathcal{P}}^{opt}$ from Eq. (25). So the theorem follows. □

Definition 3.8: CPU workload bound with respect to the optimal solution : Given a set of transactions $\mathcal{T} = \{\tau_1, \tau_2, .., \tau_m\}$ and its optimal CPU workload $U_\mathcal{T}^{opt}$, the CPU workload bound of any solution $\mathcal{S}$ with respect to its optimal solution, $\mathcal{B}_\mathcal{S}$, is defined as

$$\mathcal{B}_\mathcal{S} = U_\mathcal{S} - U_\mathcal{T}^{opt}, \tag{27}$$

where $U_\mathcal{S}$ is the CPU workload of solution $\mathcal{S}$.

Theorem 3.10: Given a set of transactions $\mathcal{T} = \{\tau_1, \tau_2, .., \tau_m\}$ with $\alpha_1 \leq \alpha_2 \leq ... \leq \alpha_m$, suppose that $\mathcal{T}$ satisfies Restriction (1) but not Restriction (2). Let $\mathcal{S}^*$ be a solution from the *SVF* algorithm. Assume that Ψ is a set of subscripts of all the transactions in $\mathcal{T}$ that are partitioned in a *partition-transformation* after which the resulting set of transactions $\mathcal{T}_\mathcal{P}$ satisfies Restriction (2). The CPU workload bound of $\mathcal{S}^*$ with respect to the optimal solution of $\mathcal{T}$, $\mathcal{B}_{\mathcal{S}^*}$, satisfies

$$\mathcal{B}_{\mathcal{S}^*} < 2 \sum_{k \in \Psi} (\frac{C_k}{\alpha_k})^2. \tag{28}$$

Proof: Let $U_{\mathcal{T}}^{opt}$, $U_{\mathcal{T}_\mathcal{P}}^{opt}$, and $U_{\mathcal{S}^*}$ denote the CPU workload of an optimal solution of $\mathcal{T}$, the optimal solution of $\mathcal{T}_\mathcal{P}$ partition-transformed from $\mathcal{T}$, and the solution $\mathcal{S}^*$ from the *SVF* algorithm, respectively. It follows from Theorem 3.9 that

$$\begin{aligned}\mathcal{B}_{\mathcal{S}^*} &= U_{\mathcal{S}^*} - U_{\mathcal{T}}^{opt} \\ &\leq U_{\mathcal{S}^*} - U_{\mathcal{T}_\mathcal{P}}^{opt}\end{aligned}$$

We prove the theorem in two steps:

1. Assume $\mathcal{T}_\mathcal{P} = \{\tau_1^*, \tau_2^*, .., \tau_{m_p}^*\}$ with $\alpha_1^* \leq \alpha_2^* \leq ... \leq \alpha_{m_p}^*$. Without loss of generality, assume $\exists k \in \Psi$ and $\tau_k \in \mathcal{T}$ is the lowest priority transaction that has been partitioned into n_k subtransactions $\tau_{k1}^*, \tau_{k2}^*, .., \tau_{kn_k}^*$ ($\tau_{ki} \in \mathcal{T}_\mathcal{P}, i = 1..n_k$). We know that $C_{k1}^* = C_{k2}^* = .. = C_{kn_k}^* = \frac{C_k}{n_k}$ and $\alpha_k = \alpha_{k1}^* = \alpha_{k2}^* = .. = \alpha_{kn_k}^*$. When these n_k subtransactions are *merged* into one transaction $\tau_k^{'}$ with $C_k^{'} = C_k$ and $\alpha_k^{'} = \alpha_k$, $\tau_k^{'}$ inherits the priority of $\tau_{kn_k}^*$ in the assignment order. So we have a new set of transactions $\mathcal{T}_1 = \mathcal{T}_\mathcal{P} - \{\tau_{k1}^*, \tau_{k2}^*, .., \tau_{kn_k}^*\} + \{\tau_k^{'}\}$. Let $U_{\mathcal{S}_1^*}$ denote the CPU workload from the SVF solution $\mathcal{S}_1^*$ of $\mathcal{T}_1$. Then

$$\begin{aligned}
&U_{\mathcal{S}_1^*} - U_{\mathcal{T}_\mathcal{P}}^{opt} \\
&= \frac{C_k^{'}}{\alpha_k^{'} - (\sum_{i=1}^{k-1} C_i^{'}) - C_k^{'}} - \sum_{l=1}^{n_k} \frac{\frac{C_k^{'}}{n_k}}{\alpha_k^{'} - (\sum_{i=1}^{k-1} C_i^{'}) - \frac{lC_k^{'}}{n_k}} \\
&= \sum_{l=1}^{n_k} (\frac{\frac{C_k^{'}}{n_k}}{\alpha_k^{'} - (\sum_{i=1}^{k-1} C_i^{'}) - C_k^{'}} - \frac{\frac{C_k^{'}}{n_k}}{\alpha_k^{'} - (\sum_{i=1}^{k-1} C_i^{'}) - \frac{lC_k^{'}}{n_k}}) \\
&= \sum_{l=1}^{n_k} (\frac{(\frac{C_k^{'}}{n_k})^2 (n_k - l)}{(\alpha_k^{'} - (\sum_{i=1}^{k-1} C_i^{'}) - C_k^{'})(\alpha_k^{'} - (\sum_{i=1}^{k-1} C_i^{'}) - \frac{lC_k^{'}}{n_k})}) \\
&\leq \sum_{l=1}^{n_k} (\frac{(\frac{C_k^{'}}{n_k})^2 (n_k - l)}{(\alpha_k^{'} - (\sum_{i=1}^{k-1} C_i^{'}) - C_k^{'})^2}) \\
&= (\frac{C_k^{'}}{n_k})^2 \frac{1+2+..+(n_k-1)}{(\alpha_k^{'} - \sum_{i=1}^{k} C_i^{'})^2} \\
&= (\frac{C_k^{'}}{n_k})^2 \frac{n_k(n_k-1)}{2(\alpha_k^{'} - \sum_{i=1}^{k} C_i^{'})^2}
\end{aligned}$$

Because $n_k(n_k - 1) < n_k^2$, we have

$$U_{\mathcal{S}_1^*} - U_{\mathcal{T}_\mathcal{P}}^{opt} < \frac{1}{2}(\frac{C_k^{'}}{\alpha_k^{'} - \sum_{i=1}^{k} C_i^{'}})^2$$

Because $\alpha_k^{'} - \sum_{i=1}^{k} C_i^{'} = P_k^{'}$, we have

$$U_{\mathcal{S}_1^*} - U_{\mathcal{T}_\mathcal{P}}^{opt} < \frac{1}{2}(\frac{C_k^{'}}{P_k^{'}})^2.$$

Since $C_k^{'} = C_k$ and $P_k^{'} \geq \frac{\alpha_k}{2}$, we have

$$U_{\mathcal{S}_1^*} - U_{\mathcal{T}_\mathcal{P}}^{opt} < 2(\frac{C_k}{\alpha_k})^2 \tag{29}$$

2. With new solution $\mathcal{S}_1^*$ and all the transactions in $\mathcal{T}_1$, let $\Psi^{'} = \Psi - \{k\}$. Similarly, for the lowest priority transaction $\tau_i \in \mathcal{T}_1$ ($i \in \Psi^{'}$), we can merge all the subtransactions in $\mathcal{T}_1$ that are partitioned from τ_i into one transaction as what

we have done for τ_k in step 1. This results in another new set of transactions, $\mathcal{T}_2$. Let $U_{\mathcal{S}_2^*}$ denote the CPU workload of $\mathcal{S}_2^*$, a SVF solution from $\mathcal{T}_2$, we have

$$U_{\mathcal{S}_2^*} - U_{\mathcal{S}_1^*} < 2(\frac{C_i}{\alpha_i})^2 \tag{30}$$

Assume there are n transactions in Ψ, repeat these steps for all the transactions in Ψ. We have

$$\begin{cases} U_{\mathcal{S}_1^*} - U_{\mathcal{T}_\mathcal{P}}^{opt} < 2(\frac{C_k}{\alpha_k})^2, k \in \Psi \\ U_{\mathcal{S}_2^*} - U_{\mathcal{S}_1^*} < 2(\frac{C_i}{\alpha_i})^2, i \in \Psi \\ ... \\ U_{\mathcal{S}_n^*} - U_{\mathcal{S}_{n-1}^*} < 2(\frac{C_j}{\alpha_j})^2, j \in \Psi \end{cases} \tag{31}$$

It is obvious that $\mathcal{S}_n^*$ is S^* of the approximate solution. Thus, for all transactions with subscripts in Ψ, we have

$\mathcal{B}_{S^*} \leq U_{S^*} - U_{\mathcal{T}_\mathcal{P}}^{opt} < \sum_{k\in\Psi} 2(\frac{C_k}{\alpha_k})^2 = 2\sum_{k\in\Psi}(\frac{C_k}{\alpha_k})^2$. □

Theorem 3.10 says that the CPU workload from SVF is within $2\sum_{k\in\Psi}(\frac{C_k}{\alpha_k})^2$ of that of an optimal solution if Restriction (1) holds. In many real applications, e.g., the avionics application [6] discussed later in Section 4, sensor transaction computation time is in the range of milliseconds, validity interval length is in the range of hundreds of milliseconds and seconds. Thus $(\frac{C_k}{\alpha_k})^2$ for a sensor transaction τ_k is about $\frac{1}{10^6}$ to $\frac{1}{10^4}$. The number of transactions which may belong to the transaction set Ψ is usually very limited. Therefore, this bound is actually very small and can be ignored in many situations, thus SVF becomes a near optimal solution.

The optimal solution for the general case of *More-Less*, i.e., when both Restrictions (1) and (2) are relaxed, is left as an open issue. However, as we shall show in Section 5, SVF is a good heuristic solution even in these situations.

3.7 Discussion of Arbitrary Jitter Bounds

In this case, the jitter bound δ_i ($\delta_i \geq 0$) of transaction τ_i ($1 \leq i \leq m$) can have an arbitrary value. The problem can be transformed to one with the same jitter bound by replacing δ with $max_{\{1\leq i\leq m\}}\{\delta_i\}$. Similarly, by replacing δ in Algorithm 3.1 with $max_{\{1\leq i\leq m\}}\{\delta_i\}$, a transaction set with deadlines and periods derived from Algorithm 3.1 is still schedulable by the deadline monotonic scheduling algorithm. More investigation is necessary for better solutions in this case. This is left for future work.

4 Application of *More-Less*: Similarity-Based Load Adjustment

In this section, we consider the similarity-based load adjustment [6] as an application of *More-Less*. The basic idea of similarity-based load adjustment is to skip the executions of transaction instances which produce similar outputs. The approach taken in [6] is to modify the execution frequencies of transactions such that only one instance of a transaction is executed for multiple periods. As a result, the system workload is reduced. View *r-serializability* [6] is a criterion used to justify the correctness of transactions. Readers are referred to [4, 6] for details of *similarity* and view *r-serializability*.

In similarity-based load adjustment, a *similarity bound* is derived for each data object based on application semantics. Two write events of the same data objects are *similar* if their sampling times differ by an amount of time no greater than the

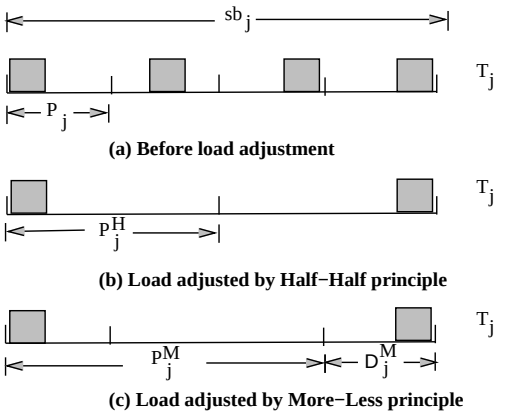

Figure 5. Update principles

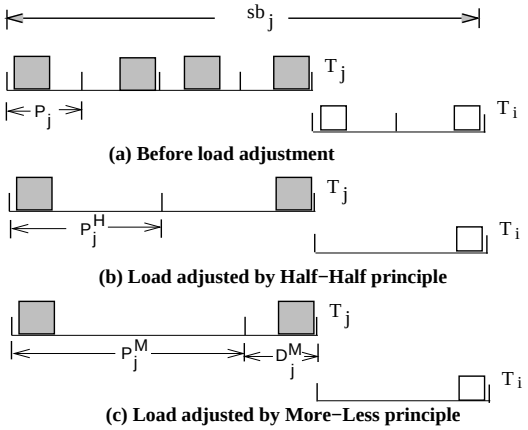

Figure 6. View principles

similarity bound. In other words, write events on the same data occurring within the similarity bound are interchangeable as input to a read without adverse effects. Therefore, some write or read events can be dropped in order to reduce system load without affecting data temporal correctness. Here, validity interval length is replaced by similarity bound to constrain the arrival time of an update transaction instance and finishing time of its next instance.

Update and *View* principles are proposed in [6] to adjust the system load. Their *update* principle is based on the *Half-Half* approach. Based on *More-Less*, we derive new *update* and *view* principles to reduce the system load even further.

Suppose sb_j is the similarity bound for data object X_j. For any two conflicting write events on X_j occurring within sb_j, they are interchangeable as input to a read event. Let P_j denote the period of transaction τ_j refreshing X_j before load adjustment. Let P_j^H and P_j^M denote the period of transaction τ_j refreshing X_j after load adjusted by *Half-Half* and *More-Less*, respectively. Also let D_j^H and D_j^M be the deadline of transaction τ_j after load adjusted by the *Half-Half* and *More-Less* approaches, respectively.

Update Principle: $P_j^M + D_j^M \leq sb_j$

In [6], the *Half-Half* approach is used to derive their *update* principle, which is $2P_j^H \leq sb_j$. However, our *update* principle derived from *More-Less* is $P_j^M + D_j^M \leq sb_j$. As shown in Figure 5, any read event will read from similar write events in both cases after load adjustment. In addition, because $D_j^M \leq \frac{sb_j}{2}$, we know that $P_j^M \geq \frac{sb_j}{2} \geq P_j^H$, which reduces the system

i	C_i	P_i	**Case 1**			**Case 2**			**Case 3**		
			P_i^M	D_i^M	P_i^H	P_i^M	D_i^M	P_i^H	P_i^M	D_i^M	P_i^H
1	1	3	12	3	6	9	3	6	3	3	3
2	2	5	5	5	5	10	10	10	15	15	15

Table 8. Parameters and results of Example 4.1

workload factor for τ_i by an amount of $\frac{C_j}{P_j^H} - \frac{C_j}{P_j^M}$ compared to the previous update principle. Therefore, the update principle derived from *More-Less* reduces load even further without sacrificing similarity-based data correctness.

View Principle: $P_j^M + D_j^M + P_i^M \leq sb_j$

Suppose transaction τ_i with period P_i reads data object X_j. Let P_i^H and P_i^M denote the period of transaction τ_i adjusted by *Half-Half* and *More-Less*, respectively. The *View* principle in [6] is defined as $2P_j^H + P_i^H \leq sb_j$. In contrast, our *view* principle from *More-Less* is defined as $P_j^M + D_j^M + P_i^M \leq sb_j$. Because $P_j^M + D_j^M = 2P_j^H$, we know $P_i^H = P_i^M$, i.e., periods of read transaction τ_i adjusted by *Half-Half* and *More-Less* are the same. As shown in Figure 6, $P_j^M + D_j^M + P_i^M$ is the maximum temporal distance among the write events which might be read by instances of τ_i and their representatives before and after load adjustment. Therefore, the *view* principle derived from the *More-Less* approach can guarantee similarity-based data correctness.

The following example clearly indicates that the *update* and *view* principles derived from *More-Less* can reduce system load more than the *update* and *view* principles from *Half-Half*.

Example 4.1: We use an example in [6] to illustrate the effectiveness of the *More-Less* approach. Suppose there are two periodic transactions τ_1 and τ_2 in a single processor environment. Their computation times and periods are given in Table 8. τ_1 periodically refreshes a data object X and τ_2 periodically reads the same data. The similarity bound sb_X of X is 22. According to *update* and *view* principles corresponding to the *More-Less* and *Half-Half* approaches, the following inequalities must hold, respectively.

$$\begin{cases} P_1^M + D_1^M \leq 22 \\ P_1^M + D_1^M + P_2^M \leq 22 \end{cases} \tag{32}$$

$$\begin{cases} 2P_1^H \leq 22 \\ 2P_1^H + P_2^H \leq 22 \end{cases} \tag{33}$$

It is obvious that there are multiple solutions. Three different results after load adjustment are shown in Table 8. Let U_H and U_M denote the system CPU workload after load adjustment based on the *Half-Half* and *More-Less* approaches, respectively. $U_H - U_M$, the difference in adjusted system load, is $\frac{1}{12}$ and $\frac{1}{18}$ in Cases 1 and 2, respectively. In Case 3, $P_2^H = P_2^M = 15$, the system load adjusted from both approaches are the same. This indicates that our update principle provides solutions with lower CPU workload than the previous update principle. □

Parameter Class	Parameters	Meaning
System	N_{CPU}	No. of CPU
	N_T	No. of temporal data groups
	N_{NT}	No. of non-temporal data
	α_i (ms)	Validity length of each temporal data groups
Sensor Trans.	C_i (ms)	CPU time for updating a temporal data group
	Length	No. of temporal data groups updated
Triggered Trans.	$CPUTime$ (ms)	CPU Time for each data access
	Length	No. of data accessed
	Slack Factor	Triggered transaction slack factor
	P_T	Probability to access a temporal data
	P_W	Write probability for a non-temporal data access

Table 9. Experimental parameters.

5 Experiments

We begin this section with the experimental setup and the assumptions made in our experiments. We also present a table of important parameters and their values. Finally, we describe each set of experiments and an analysis of the results. All our experiments are conducted for a main memory database setting. The primary performance metric we use in our paper is Missed Deadline Ratio (MDR), which is a traditional metric used to evaluate performance in real-time database systems. Let N_{miss} denote the number of transactions that miss deadlines, and $N_{succeed}$ denote the number of transactions that succeed. The MDR is given by the following expression:

$$MDR = \frac{N_{miss}}{N_{miss} + N_{succeed}}$$

In our paper, a transaction is aborted as soon as its deadline expires. This corresponds to a firm real-time transaction. This policy assumes that finishing a transaction after its deadline expires does not impart any value to the system.

A performance model of a real-time database, $RADEx$ [14], was developed for our experimental studies. In the following, only statistically different results are discussed.

5.1 Simulation Model and Parameters

In this section, two sets of experimental results are presented to quantitatively compare *More-Less* and *Half-Half*. In the first set of experiments, it is shown that *More-Less* produces solutions with better schedulability and lower CPU workload than the *Half-Half* approach. In the second set of experiments, mixed transaction workloads are scheduled: a class of sensor update transactions that maintain the validity of temporal data, and a class of transactions that are triggered by the updates of sensor transactions. Triggered transactions access both temporal and non-temporal data. Sensor transactions are periodic transactions scheduled by the deadline monotonic scheduling algorithm. Triggered transactions are aperiodic

Parameter Class	Parameters	Value
System	N_{CPU}	1
	N_T	$100 - 350$
	N_{NT}	1000
	α_i (ms)	$[4000, 8000]$
Sensor Trans.	C_i (ms)	$[5, 15]$
	Length	1
Triggered Trans.	$CPUTime$ (ms)	5
	Length	$[5, 15]$
	Slack Factor	8
	P_T	0.5
	P_W	0.0

Table 10. Experimental settings.

transactions scheduled by the earliest deadline first scheduling algorithm. Given transactions belonging to different classes, sensor transactions are given higher priorities than triggered transactions. For simplicity of simulation, only one version of temporal data is maintained. Upon refreshing a temporal data, the older version is discarded. A triggered transaction always copies a temporal data into its local working area before it reads the data. Thus concurrency control for temporal data is not considered. A triggered transaction that cannot commit before the validity of any temporal data read by it expires has to be aborted, and restarted later if it has not missed its deadline. In such a case, a data-deadline [14] is imposed on the triggered transaction due to the temporal constraints resulting from data validities. 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 [14].

A summary of the parameters and default settings used in experiments are presented in Tables 9 and 10. The values are similar to the values used in the experiments of [6] and data presented in the study of air traffic control system in [8]. Three classes of parameters are presented: system parameters, sensor transaction parameters and triggered transaction parameters. For system configurations, we only consider a system with a single CPU. The number of non-temporal data is fixed at 1000, while the number of temporal data groups is varied from 100 to 350. It is assumed that each group of temporal data has the same validity interval length, and validity interval length of each group is uniformly varied from 4000 to 8000 ms. For sensor transactions, it is assumed that each sensor transaction updates one group of temporal data, and the CPU time for each transaction is uniformly varied from 5 to 15 ms. For triggered transactions, it is assumed that the number of data accessed by each transaction is uniformly varied from 5 to 15, while each data access takes 5 ms of CPU time. The slack factor determines the slack of a transaction before its deadline expires. The slack factor of each triggered transaction 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:

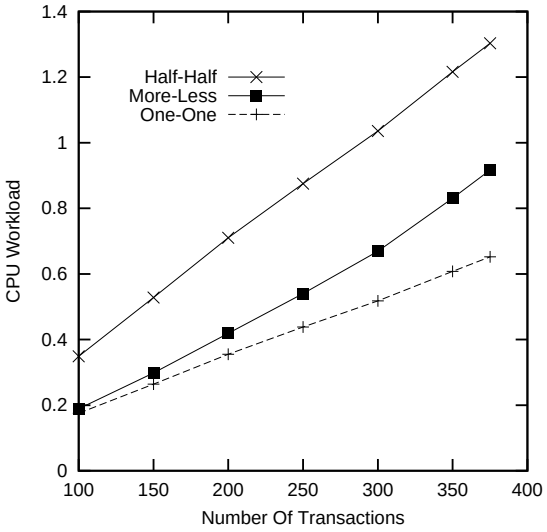

Figure 7. CPU workloads from three approaches

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

For each data access of a triggered transaction, the probability to access a temporal data is P_T. For each non-temporal data access of a triggered transaction, the probability to update that data is determined by P_W.

5.2 Experiment 1: Comparison of CPU Workloads

In the first set of experiments, the CPU workloads of sensor update transactions with deadlines and periods produced by *More-Less* and *Half-Half* are quantitatively compared. A set of sensor transactions is generated randomly: computation time of a sensor transaction is uniformly generated from 5 to 15 milliseconds, and the validity interval length of a temporal data group is uniformly generated from 4000 to 8000 milliseconds. The number of sensor transactions are varied to change the workload in the system.

The resulting CPU workload generated from the *One-One*, *Half-Half* and *More-Less* with SVF ordering are presented in Figure 7. When the number of transactions is less than 200, the CPU workload falls into the *restricted case*, i.e., Restriction (1) is satisfied. This is because the sum of computation times of all the transactions is less than half of the minimum of all the validity interval lengths. It is observed that the CPU workload produced by *More-Less* is very close to that of *One-One*, and much less than that of the *Half-Half* approach. We would like to remind readers that *One-One* is used only as an artificial baseline – it *does not* guarantee the validity of temporal data. In this case, as we explained in Section 3.6, *More-Less* is very close to the optimal solution. This is clearly substantiated by the small difference in the CPU workload between *One-One* and *More-Less*: the CPU workload of an optimal solution under *More-Less* should be between those for *One-One* and *More-Less* with SVF ordering. When the number of transactions is more than 200, the CPU workload falls into the *general case* because Restriction (1) is *not* satisfied. In this case, we observe that CPU workload of *More-Less* is still much less than that of *Half-Half*. However, the difference in CPU workload of *One-One* and *More-Less* increases as system workload increases. The highest workload in our experiments is produced when the number of transactions is 375, and the corresponding CPU workload under *One-One*, *Half-Half* and *More-Less* is about 65%, 130% and 92%, respectively. *Half-Half* cannot produce a feasible solution when the number of transactions exceeds 300 because the corresponding CPU workload exceeds 100%.

But *More-Less* can still produce feasible solutions even when the number of transactions increases to 375.

In summary, when both the *Half-Half* and *More-Less* approaches can be used to schedule a set of sensor update transactions, the *More-Less* approach can be used to produce solutions with much lower CPU workload, thus more CPU capacity can be used by other transactions in the system. In addition, *More-Less* can be used to provide feasible solutions even when *Half-Half* cannot be applied. In such situations, *More-Less* provides better *schedulability*.

5.3 Experiment 2: Co-scheduling of Mixed Transaction Workloads

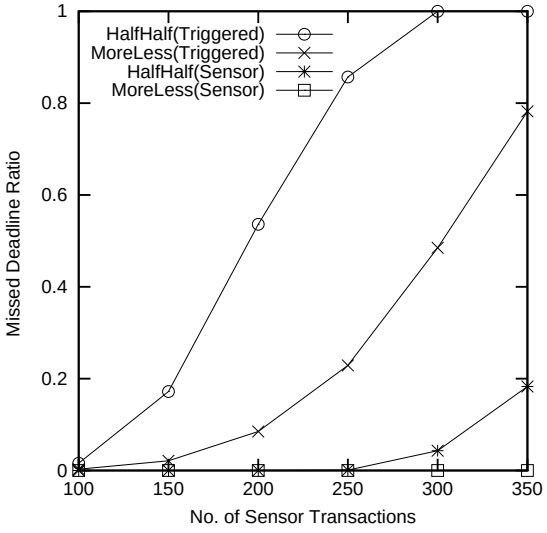

Figure 8. MDR comparison with fixed arrival rate of triggered transactions

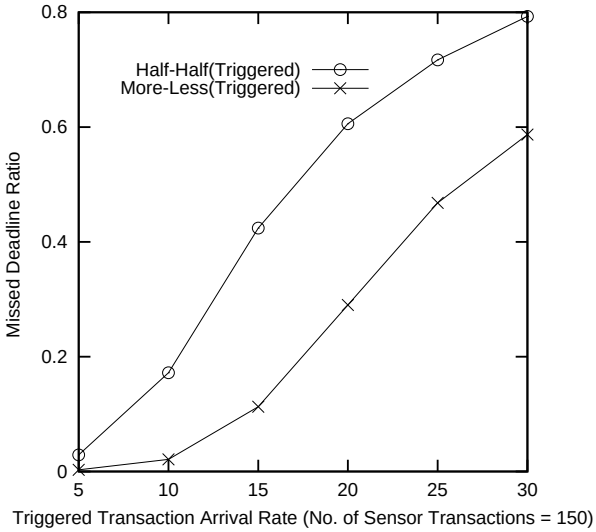

Figure 9. MDR comparison with fixed number of sensor transactions

In this set of experiments, the quantitative performances of sensor transactions and triggered transactions under Half-Half and More-Less are compared. We conduct experiments in two scenarios: (1) the arrival rate of triggered transactions is fixed, and (2) the number of sensor transactions is fixed.

Figure 8 presents the missed deadline ratio of sensor transactions and triggered transactions under Half-Half and More-Less, respectively, while the arrival rate of triggered transactions is fixed at 10 transactions per second. The number of sensor update transactions is gradually increased from 100 to 350. As observed from Figure 8, some of the sensor transactions under Half-Half miss their deadlines when the number of sensor transactions is greater than or equal to 300. This is because the CPU workload resulting from sensor transactions already exceeds CPU capacity, as shown in Figure 7. However, none of the sensor transactions under More-Less miss their deadlines even when the number of sensor transactions is 350. This clearly demonstrates that More-Less provides better schedulability than Half-Half. Furthermore, the missed deadline ratio of triggered transactions under Half-Half is much higher than that under More-Less. For example, when the number of sensor transactions is 200, only 10% triggered transactions miss their deadlines under More-Less, but more than 50% triggered transactions miss their deadlines under Half-Half. This is because the CPU workload of sensor transactions under Half-Half is much higher than that under More-Less. In Figure 8, it is also observed that all of the triggered transactions miss their deadlines under Half-Half when the number of sensor transactions is 300. In this case, as mentioned earlier, the CPU workload resulting from sensor transactions already exceeds CPU capacity. Since the class of sensor transactions is assigned higher priority than the class of triggered transactions, none of the triggered transactions can complete before their deadlines.

Figure 9 presents the missed deadline ratio of triggered transactions under Half-Half and More-Less, respectively, while the number of sensor transactions is fixed at 150. The arrival rate of triggered transactions is gradually increased from 5 to 30 transactions per second. As observed from Figure 9, the missed deadline ratio of triggered transactions under More-Less is much lower than that under Half-Half. For example, when the arrival rate of triggered transactions is 15 transactions per second, more than 40% triggered transactions miss their deadlines under Half-Half, whereas only 10% triggered transactions miss their deadlines under More-Less with the same arrival rate of triggered transactions. This also results from the lower CPU workload of sensor transactions produced by More-Less.

In summary, More-Less also provides better performance in mixed workloads scenario. In particular, other workloads can benefit from CPU workload reduction of sensor transactions by using More-Less.

6 Related Work

Database systems in which time validity intervals are associated with the data are discussed in [14, 13, 6, 5, 4, 2, 11, 12]. Such systems introduce the need to maintain data temporal consistency in addition to logical consistency.

In the model introduced in [13], 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 [13].

In [4, 5], 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.

Data-deadline is proposed in our previous work [14]. We proposed data-deadline based scheduling, forced-wait and similarity based scheduling policies to maintain temporal consistency of real-time data and meet transaction deadlines in

RTDBs.

A design methodology for guaranteeing end-to-end requirements of real-time systems is presented in [2]. Their approach guarantees end-to-end propagation delay, temporal input-sampling correlation, and allowable separation times between updated output values. However, their solution is based on the assumption that all the periodic tasks have harmonic periods. However, we do not make the assumption that all the periods are harmonic.

The work presented in our paper is also related to the work of [6]. But, as we showed, the schedulability of *More-Less* is better than *Half-Half* used in [6]. It is noted that *More-Less* guarantees a bound on the arrival time of a periodic transaction instance and the finishing time of the next instance. This is different from the *distance constrained scheduling*, a *dynamic* scheduling mechanism, which guarantees a bound of the finishing times of two consecutive instances of a task [3]. *Distance constrained scheduling* is also used in [16] to provide temporal consistency guarantees for real-time primary-backup replication service.

Very recently, we came across a paper by Burns and Davis [1] where what we refer to as SVF is proposed as a heuristic to determine periods. As we show in this paper, SVF in fact provides an optimal task assignment order when Restrictions (1) and (2) are met and is a tight approximate ordering criterion when only Restriction (1) is met.

7 Conclusions

In this paper, we examined the problem of deadline and period assignment in systems where data freshness should be guaranteed. *More-Less*, a novel approach based on the *validity constraint*, *deadline constraint* and *schedulability constraint* is proposed and analyzed. The solution for *More-Less* is constructed according to the *deadline monotonic* scheduling algorithm, which is the best algorithm for fixed priority scheduling. We proved the correctness of the *More-Less* approach, and its superiority to the traditional approach, the *Half-Half* approach. We further examined the issue of optimal assignment order under *More-Less* approach and found that *Shortest Validity First (SVF)* is an optimal order in situations in which both Restrictions (1) and (2) hold. With the relaxation of Restriction (2), we proved that SVF is an approximate solution within a certain bound of the optimal solutions. We showed, through analysis, that this bound is tight in real world applications. We have also found in experiments that *More-Less* with the SVF assignment order produces solutions with much better schedulability as well as lower CPU workload than *Half-Half* even in general cases, i.e., when Restriction (1) does not hold. However, the problem of searching for optimal assignment orders in the general case remains open.

Our experimental results demonstrate that More-Less is a very effective approach for scheduling sensor transaction workloads with validity constraints. It is also demonstrated that More-Less can also provide better performance for triggered transactions by reducing the CPU workloads of sensor transactions. Because of its simplicity and effectiveness, the More-Less approach can be applied in practical real-time applications.

Acknowledgement

Authors thank Jayant Haritsa for discussions of jitters.

References

[1] 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.

[2] R. Gerber, S. Hong and M. Saksena, "Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes," *IEEE 15th Real-Time Systems Symposium*, December 1994.

[3] 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.

[4] T. Kuo and A. K. Mok, "Real-Time Data Semantics and Similarity-Based Concurrency Control," *IEEE 13th Real-Time Systems Symposium* , December 1992.

[5] T. Kuo and A. K. Mok, "SSP: a Semantics-Based Protocol for Real-Time Data Access," *IEEE 14th Real-Time Systems Symposium* , December 1993.

[6] S. Ho, T. Kuo, and A. K. Mok, "Similarity-Based Load Adjustment for Static Real-Time Transaction Systems," *18th Real-Time Systems Symposium*, 1997.

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

[8] Doug 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.

[9] J. Lehoczky, L. Sha and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE 10th Real-Time Systems Symposium*, 1989.

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

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

[12] 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.

[13] 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.

[14] M. Xiong, R. M. Sivasankaran, J. Stankovic, K. Ramamritham, and D. Towsley, "Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics," *IEEE 17th Real-Time Systems Symposium*, pp. 240-251, December 1996.

[15] M. Xiong and K. Ramamritham, "Deriving Deadlines and Periods for Real-Time Update Transactions," in *Proc. 20th IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December, 1999.

[16] H. Zhou, and F. Jahanian, "Real-Time Primary-Backup (RTPB) Replication with Temporal Consistency Guarantees," *Proceedings of ICDCS*, May, 1998.

8 Appendix

To prove Theorem 3.6, we first introduce the following lemma.

Lemma 8.1: Suppose $\frac{\alpha_j}{2} \geq \sum_{i=1}^{2} C_i$ $(1 \leq j \leq 2)$. Eq. (18) is satisfied iff

$$(\alpha_1 - C_1)^2 - (\alpha_2 - C_2)^2 \leq C_2(\alpha_1 - C_1) - C_1(\alpha_2 - C_2). \tag{34}$$

Proof: $\frac{C_1}{\alpha_1 - C_1} + \frac{C_2}{\alpha_2 - (C_1 + C_2)} \leq \frac{C_2}{\alpha_2 - C_2} + \frac{C_1}{\alpha_1 - (C_1 + C_2)}$

$\Leftrightarrow \frac{C_2}{\alpha_2 - (C_1 + C_2)} - \frac{C_2}{\alpha_2 - C_2} \leq \frac{C_1}{\alpha_1 - (C_1 + C_2)} - \frac{C_1}{\alpha_1 - C_1}$

$\Leftrightarrow (\alpha_1 - C_1)(\alpha_1 - C_1 - C_2) \leq (\alpha_2 - C_2)(\alpha_2 - C_1 - C_2)$

$\Leftrightarrow (\alpha_1 - C_1)^2 - C_2(\alpha_1 - C_1) \leq (\alpha_2 - C_2)^2 - C_1(\alpha_2 - C_2)$

$\Leftrightarrow (\alpha_1 - C_1)^2 - (\alpha_2 - C_2)^2 \leq C_2(\alpha_1 - C_1) - C_1(\alpha_2 - C_2)$. □

Proof of Theorem 3.6: $\Delta L_{ij} = (\alpha_i - C_i) - (\alpha_j - C_j) \Leftrightarrow (\alpha_i - C_i) = (\alpha_j - C_j) + \Delta L_{ij}$.

Thus,

$$C_j = C_i + \Delta L_{ij} + \Delta\alpha_{ji}. \tag{35}$$

Let $K = \alpha_i - (C_i + C_j)$, combined with Eq. (35), we have $\alpha_j - C_j = C_i + K + \Delta\alpha_{ji}$.

We know that

$(\alpha_i - C_i)^2 - (\alpha_j - C_j)^2$

$= ((\alpha_j - C_j) + \Delta L_{ij})(\alpha_i - C_i) - (\alpha_j - C_j)^2$

$= (\alpha_j - C_j)((\alpha_i - C_i) - (\alpha_j - C_j)) + \Delta L_{ij}(\alpha_i - C_i)$

$= (C_i + K + \Delta\alpha_{ji})((\alpha_i - C_i) - (\alpha_j - C_j)) + \Delta L_{ij}(\alpha_i - C_i)$

$= (C_i + \Delta\alpha_{ji} + \Delta L_{ij})(\alpha_i - C_i) - C_i(\alpha_j - C_j) + K\Delta L_{ij} - \Delta\alpha_{ji}(\alpha_j - C_j)$

$= C_j(\alpha_i - C_i) - C_i(\alpha_j - C_j) + (\alpha_i - (C_i + C_j))\Delta L_{ij} - \Delta\alpha_{ji}(\alpha_j - C_j)$.

Now we have $(\alpha_i - C_i)^2 - (\alpha_j - C_j)^2 =$

$$C_j(\alpha_i - C_i) - C_i(\alpha_j - C_j) + (\alpha_i - (C_i + C_j))\Delta L_{ij} - \Delta\alpha_{ji}(\alpha_j - C_j). \tag{36}$$

By Lemma 8.1, it suffices to show that

$(\alpha_i - C_i)^2 - (\alpha_j - C_j)^2 \leq C_j(\alpha_i - C_i) - C_i(\alpha_j - C_j)$, i.e., we want to prove that $(\alpha_i - (C_i + C_j))\Delta L_{ij} - \Delta\alpha_{ji}(\alpha_j - C_j) \leq 0$ from Eq. (36).

Since $\Delta\alpha_{ji} \geq 0$, we know $\alpha_i \leq \alpha_j$. It is true that

$$0 < \frac{\alpha_i - (C_i + C_j)}{\alpha_j - C_j} < 1 \tag{37}$$

- If $\Delta L_{ij} \leq 0$, then $\frac{\alpha_i - (C_i + C_j)}{\alpha_j - C_j}\Delta L_{ij} \leq 0$. We have $\Delta\alpha_{ij} \geq 0 \geq \frac{\alpha_i - (C_i + C_j)}{\alpha_j - C_j}\Delta L_{ij}$, i.e., $(\alpha_i - (C_i + C_j))\Delta L_{ij} - \Delta\alpha_{ji}(\alpha_j - C_j) \leq 0$.
- If $\Delta L_{ij} > 0$, because $\Delta C_{ji} \leq 2\Delta\alpha_{ji} \Leftrightarrow (\alpha_i - \alpha_j) + (C_j - C_i) \leq \Delta\alpha_{ji}$, we have

  $$\Delta L_{ij} \leq \Delta\alpha_{ji}. \tag{38}$$

  Combining Eq. (38) and Eq. (37), we have $\Delta\alpha_{ji} \geq \Delta L_{ij} > \frac{\alpha_i - (C_i + C_j)}{\alpha_j - C_j}\Delta L_{ij}$. Therefore, we know $\Delta\alpha_{ji} > \frac{\alpha_i - (C_i + C_j)}{\alpha_j - C_j}\Delta L_{ij}$, i.e., $(\alpha_i - (C_i + C_j))\Delta L_{ij} - \Delta\alpha_{ji}(\alpha_j - C_j) < 0$.

Therefore we have

$$(\alpha_i - C_i)^2 - (\alpha_j - C_j)^2 \leq C_j(\alpha_i - C_i) - C_i(\alpha_j - C_j). \tag{39}$$

From Lemma 8.1, we know that $\frac{C_i}{\alpha_i - C_i} + \frac{C_j}{\alpha_j - (C_i + C_j)} \leq \frac{C_j}{\alpha_j - C_j} + \frac{C_i}{\alpha_i - (C_i + C_j)}$. □