

A Schedulability Analysis of Deferrable Scheduling Using Patterns

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

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

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

Aloysius K. Mok
The University of Texas at Austin
mok@cs.utexas.edu

Abstract

The schedulability testing for the deferrable scheduling algorithm for fixed priority transactions (DS-FP) remains an open problem since its introduction. In this paper, we take the first step towards investigating necessary and sufficient conditions for the DS-FP schedulability. We propose a necessary and sufficient schedulability condition for the algorithm in discrete time systems, and prove its correctness. Based on this condition, we propose a schedulability test algorithm that is more accurate than the existing test that is only based on a sufficient condition. Our algorithm exploits the fact that there is always a repeating pattern in a DS-FP schedule in discrete time systems. We demonstrate through examples that our schedulability test algorithm outperforms the existing algorithm in terms of accuracy.

1 Introduction

The quality and timeliness of data values sampled from real-world entities have been critical for real-time data services. These sampled data, usually called real-time data, could become invalid with the passage of time since the status of the corresponding entity in the real-world may change continuously. Sensor update transactions (or update transactions for short) need to constantly sample real-world data values and install them into Real-Time Databases (RTDBS). Otherwise, the system cannot detect and respond to environmental changes in a timely manner.

One efficient way to determine the correctness of real-time data is to define a *validity constraint* [20, 18, 11, 12, 13, 5, 22, 10, 14, 7, 8, 3, 23, 24] or *age constraint* [2], which determines a *validity interval* length for each real-time data object. A real-time data value is only valid within the *validity interval*. In practice, for safety and reliability reasons, RTDBSs require continuous generation of update transactions to refresh real-time data objects regardless of how much the status of the corresponding real-world entities have been changed. To meet this constraint, it is impor-

tant to produce a schedule for all update transactions such that for any consecutive updates of a real-time data object, the next update is completed before the previous data version expires. It is crucial to schedule update transactions efficiently in the design of real-time applications.

Most of the work in update transaction scheduling assumes a periodic transaction model. For example, a periodic update transaction model is adopted in the more-less (*ML*) scheme [23, 14, 3]. *ML* can guarantee the validity of real-time data objects. Recently, the deferrable scheduling algorithm for fixed priority transactions (*DS-FP*) was proposed [24] to reduce the total update transaction workload. The intuition of *DS-FP* is to schedule an update transaction job as late as possible based on the sampling time of its previous job.

DS-FP reduces update workload by sacrificing the periodicity of the update transactions. This poses great challenges for its schedulability analysis. One prominent method in classical schedulability analysis is based on the critical instant test. A critical instant makes sense for periodic tasks. Also sporadic task sets are usually reduced to periodic ones for their schedulability analysis. For example, the minimum separation times are treated as periods in the schedulability analysis of sporadic tasks. In a *DS-FP* schedule, however, the distance of the release times of two consecutive jobs is not a constant. This distance may vary from a transaction's execution time to its validity interval minus the execution time. It is only proved so far that *DS-FP* could schedule any transaction set that is schedulable by *ML* [25]. This sufficient condition has seriously restricted the schedulability of a transaction set under *DS-FP*. While experimental results have demonstrated that *DS-FP* outperforms *ML* significantly [24], the transaction sets used in the experiments are all schedulable by *ML*. *DS-FP* is demonstrated with its lower processor utilization while there exists no method to determine if a transaction set is schedulable by *DS-FP* even if it is not schedulable by *ML*. The open theoretic question is whether there is any necessary and sufficient condition to determine if a transaction

Symbol	Definition
X_i	Real-time data object i
τ_i	Update transaction updating X_i
$J_{i,j}$	The j th job of τ_i ($j = 0, 1, 2, \dots$)
$R_{i,j}$	Response time of $J_{i,j}$
C_i	Execution time of transaction τ_i
\mathcal{V}_i	Validity (interval) length of X_i
$f_{i,j}$	Finishing time of $J_{i,j}$
$r_{i,j}$	Release (Sampling) time of $J_{i,j}$
$d_{i,j}$	Absolute deadline of $J_{i,j}$
P_i	Period of transaction τ_i in <i>ML</i>
D_i	Relative deadline of transaction τ_i in <i>ML</i>
\mathcal{P}	A fixed pattern repeating in a <i>DS-FP</i> schedule
\mathcal{P}_s	Pattern \mathcal{P} starting time
\mathcal{P}_l	Length of pattern \mathcal{P}
$\mathcal{S}_\tau(t)$	State of transaction τ at time t
$\mathcal{S}_\mathcal{T}(t)$	State of transaction set \mathcal{T} at time t
$\Theta_i(a, b)$	Total cumulative processor demands from higher-priority transactions received by τ_i in interval $[a, b)$

Table 1. Symbols and definitions.

set is schedulable by *DS-FP* (even if it is not schedulable by *ML*).

This paper takes the first step towards addressing this important problem. We prove that there always exists a repeating pattern for a given *DS-FP* schedule in discrete time systems. This provides an important foundation to derive a necessary and sufficient condition for schedulability test of *DS-FP*. Based on this condition, a schedulability test algorithm is derived.

This paper is organized as follows: Section 2 briefly reviews the background and related work in the area. Section 3 presents the major contributions of this paper. In particular, we present the proof of our necessary and sufficient schedulability condition in discrete time systems, and our schedulability test algorithm based on this condition. We conclude the paper in Section 4.

2 Background and Related Work

This section briefly reviews the *temporal validity* of real-time data, and the *More-Less (ML)* [2, 23] and deferrable scheduling (*DS-FP*) [24] algorithms. Formal definitions of the frequently used symbols are given in Table 1.

Definition 2.1: A real-time data object (X_i) at time t is temporally valid (or absolutely consistent) if, for its j^{th} update finished latest before t , the sampling time ($r_{i,j}$) plus the *validity interval* (\mathcal{V}_i) of the data object is not less than t , i.e., $r_{i,j} + \mathcal{V}_i \geq t$ [18]. \square

A data value for real-time data object X_i sampled at any time t will be valid up to $(t + \mathcal{V}_i)$. The actual length of the *temporal validity interval* of a real-time data object is application dependent [18, 19, 17]. One of the important design goals of RTDBS is to guarantee that real-time data remain fresh, i.e., they are always valid.

Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^m$, we assume without loss of generality that τ_k has higher priority than τ_j for $k < j$. Please note that both *ML* and *DS-FP* are preemptive scheduling algorithms.

More-Less: *ML* adopts the periodic task model [15] for sensor update transactions whose derived deadlines are not larger than their periods. Consider *synchronous* transactions whose first jobs all start at time 0. A time instant after which a transaction job has the worst-case response time is called a *critical instant*, e.g., time 0 is a critical instant for all the transactions with deadlines no larger than their periods if those transactions are *synchronous* [15]. Note that we only consider synchronous transactions. In *ML*, there are three constraints to follow for transactions τ_i ($\forall i, 1 \leq i \leq m$) [23]:

- *Validity constraint:* the sum of the period and relative deadline of transaction τ_i is always less than or equal to \mathcal{V}_i , i.e.,

$$P_i + D_i \leq \mathcal{V}_i \quad (1)$$

- *Deadline constraint:* the period of an update transaction is assigned to be more than or equal to half of the validity length of its updated object, while its corresponding relative deadline is less than or equal to half of the validity length of the same object. For τ_i to be schedulable, D_i must be greater than or equal to C_i , the worst-case execution time of τ_i , i.e., $C_i \leq D_i \leq P_i$.
- *Schedulability constraint:* for a given set of update transactions, the *Deadline Monotonic* scheduling algorithm is used to schedule the transactions. Consequently, $\sum_{j=1}^i (\lceil \frac{D_i}{P_j} \rceil \cdot C_j) \leq D_i$ ($1 \leq i \leq m$).

ML assigns priorities to transactions based on *Shortest Validity First (SVF)*, i.e., in the *inverse* order of validity length, and ties are resolved in favor of transactions with larger execution time (C_i). It assigns deadlines and periods to τ_i as follows:

$$D_i = f_{i,0}^{ml} - r_{i,0}^{ml}, \quad (2)$$

$$P_i = \mathcal{V}_i - D_i, \quad (3)$$

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

DS-FP: *ML* is pessimistic on the deadline and period assignment. This is because it uses a periodic task model that has a fixed period and relative deadline for each transaction, and the relative deadline D_i is equal to the worst-case response time of the transaction. According to the *validity constraint* in *ML*, the larger the deadline D_i , the smaller the period P_i . In order to increase the separation of two consecutive jobs (and thus reduce the sensor update workload),

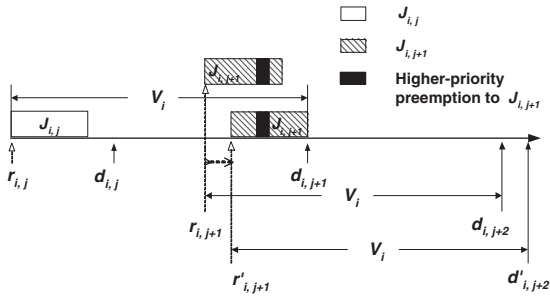


Figure 1. Illustration of DS-FP scheduling: $r_{i,j+1}$ can be shifted to $r'_{i,j+1}$ without violating the validity constraint.

DS-FP adaptively derives the relative deadline and separation of one job from its previous job and preemptions from higher-priority transactions. Given release time $r_{i,j}$ of job $J_{i,j}$ and deadline $d_{i,j+1}$ of job $J_{i,j+1}$ ($j \geq 0$),

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

guarantees that Eq. 1 can be satisfied, as depicted in Figure 1. Correspondingly, the following equation follows directly from Eq. 4:

$$(r_{i,j+1} - r_{i,j}) + (d_{i,j+1} - r_{i,j+1}) = \mathcal{V}_i. \quad (5)$$

If $r_{i,j+1}$ can be shifted onward to $r'_{i,j+1}$ along the time line in Figure 1, it does not violate Eq. 5. After the shift, temporal validity can still be guaranteed as long as $J_{i,j+1}$ is completed by its deadline $d_{i,j+1}$. The idea of DS-FP is to defer the sampling time (i.e., release time), $r_{i,j+1}$, of $J_{i,j}$'s next job as late as possible while still guarantee Eq. 1. We now introduce a definition that is used in the description of DS-FP.

Definition 2.2: Let $\Theta_i(a, b)$ denote the total *cumulative processor demands* made by all jobs of higher-priority transaction τ_j for $\forall j$ ($1 \leq j \leq i-1$) during the time interval $[a, b)$ from a schedule \mathcal{S} produced by a fixed priority scheduling algorithm. Then,

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

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

According to the fixed priority scheduling theory, $r_{i,j+1}$ in DS-FP can be derived backwards from its deadline $d_{i,j+1}$ as follows:

$$r_{i,j+1} = d_{i,j+1} - R_{i,j+1}(r_{i,j+1}, d_{i,j+1}); \quad (6)$$

$$R_{i,j+1}(r_{i,j+1}, d_{i,j+1}) = \Theta_i(r_{i,j+1}, d_{i,j+1}) + C_i. \quad (7)$$

where $R_{i,j+1}(r_{i,j+1}, d_{i,j+1})$ (or $R_{i,j+1}$ for simplicity in DS-FP) denotes the response time of $J_{i,j+1}$ deriving backwards from its deadline $d_{i,j+1}$. Note that the schedule of all higher-priority jobs that are released prior to $d_{i,j+1}$ needs to be computed before computing $\Theta_i(r_{i,j+1}, d_{i,j+1})$.

Similar to ML, DS-FP also assigns priorities to transactions according to SVF. Readers are referred to [25] for the details of the DS-FP algorithm. Now we summarize the algorithm as follows. First we set $r_{i,0} = 0, \forall i, 1 \leq i \leq m$. The highest priority job among the outstanding jobs is always scheduled first. It is only preempted when a new job with higher priority is ready. As soon as a job $J_{i,j}$ is completed, we derive the $r_{i,j+1}$ of its next job according to above calculations. The algorithm fails when a job misses its deadline. Otherwise it keeps running. It is proved that any task set that is scheduled by ML is also scheduled by DS-FP [25].

Theorem 2.1: (Corollary 3.2 in [25]) Given a synchronous update transaction set \mathcal{T} with known C_i and \mathcal{V}_i ($1 \leq i \leq m$), if \mathcal{T} can be scheduled by ML, then it can also be scheduled by DS-FP.

3 DS-FP Schedulability Analysis

It is unclear how to perform a schedulability test for DS-FP if a set of transactions cannot be scheduled by ML. This section presents a method based on DS-FP pattern analysis that can be used for DS-FP schedulability testing. In Section 3.1, our examples demonstrate that transaction sets schedulable by DS-FP are not necessarily schedulable by ML. Section 3.2 proves the existence of repeating patterns in a valid DS-FP schedule. Section 3.3 presents an algorithm that searches for the repeating pattern in a DS-FP schedule. The search algorithm leads to a schedulability test algorithm in Section 3.4. Finally Section 3.5 discusses DS-FP in Continuous Time Systems.

From here on, it is assumed that transactions are studied in a discrete time system unless it is specified otherwise. It is also assumed that the DS-FP scheduler only considers the *worst-case* execution time of a transaction in its schedule. In other words, if the execution time of a transaction job is less than the worst-case execution time of the transaction, the job can be idled in its assigned time slots after its execution is finished. This assumption simplifies our schedulability analysis.

3.1 ML vs. DS-FP

Theorem 2.1 states that DS-FP dominates ML in terms of schedulability. That is, if \mathcal{T} can be scheduled by ML, then it can also be scheduled by DS-FP. However, the converse statement is not true. This can be demonstrated in the following examples.

Example 3.1: Consider a set of two transactions $\{\tau_1, \tau_2\}$ with execution times 2 and 3, and validity intervals 6 and

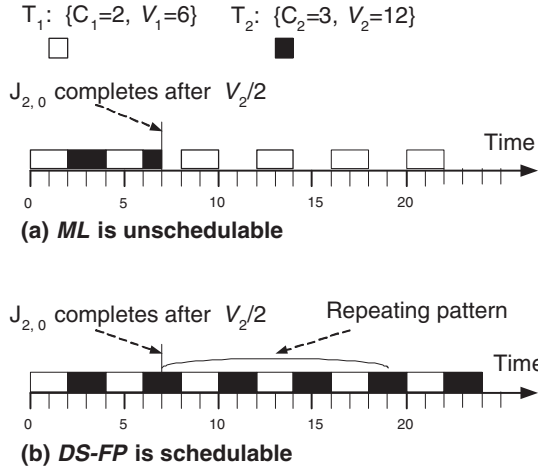


Figure 2. Two transactions that can be scheduled by *DS-FP* but not by *ML*

12 respectively. Figure 2 (a) depicts a schedule of the transactions under *ML*. The first job of τ_2 , $J_{2,0}$, completes at time 7, which is greater than $\frac{V_2}{2} = 6$. Thus the set of transactions is not schedulable by *ML*.

Figure 2 (b) depicts a schedule of the transactions under *DS-FP*. The same transaction set is schedulable by *DS-FP* because the schedule pattern between time 7 and 19 repeats itself forever. \square

DS-FP is better in Example 3.1 because *DS-FP* allows $J_{2,0}$ to be completed later than $\frac{V_2}{2}$. There are also transaction sets in which for every transaction τ_i , $J_{i,0}$ is completed no later than $\frac{V_i}{2}$ in *DS-FP*, and further more these transaction sets can be scheduled by *DS-FP* but not by *ML*. This is because *DS-FP* leaves extra processor time early on to be used by $J_{i,0}$ s of lower priority transactions. The next example illustrates this point.

Example 3.2: Consider a set of three transactions $\{\tau_1, \tau_2, \tau_3\}$ with execution times 2, 3, 3, and validity intervals 6, 15, 47, respectively. Figure 3 (a) depicts a schedule of the transactions under *ML*. The *ML* period and deadline for τ_2 are 8 and 7, respectively. The first job of τ_3 , $J_{3,0}$, completes at time 24, which is greater than $\frac{V_3}{2} = 23.5$. Thus the set of transactions is not schedulable by *ML*.

Figure 3 (b) depicts a schedule of the transactions under *DS-FP*. The same transaction set is schedulable by *DS-FP* because the schedule pattern between time 26 and 50 repeats itself forever. In this schedule $J_{3,0}$ completes at time 19, which is smaller than $\frac{V_3}{2}$ (i.e., 23.5). This is because $J_{2,2}$ is scheduled later than that of *ML*. \square

Note that *DS-FP* fully utilizes the processor in both examples. We could easily derive examples in which the pro-

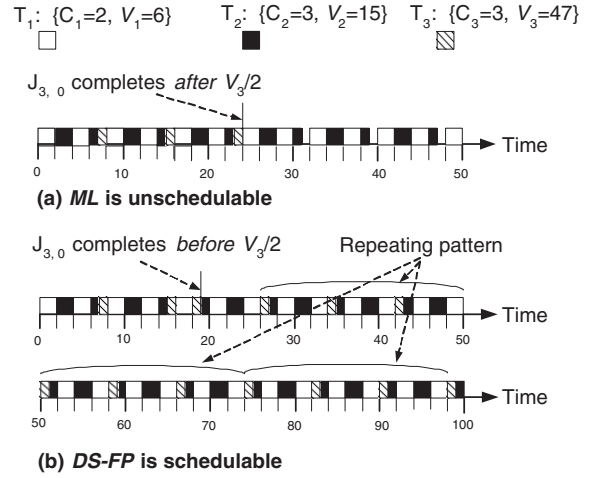


Figure 3. Three transactions that can be scheduled by *DS-FP* but not by *ML*

cessor idles once in a while. For example, in Figure 2 we could change C_2 to 2.5 and in Figure 3 we could change C_3 to 2.75. After both changes the transaction sets still cannot be scheduled by *ML* but can by *DS-FP*. Further more, we could then scale up the numbers to make them all integers again.

Note that we prove that a transaction set is schedulable by *DS-FP* by demonstrating that a repeating pattern occurs for the transaction set. The remaining questions are whether a repeating pattern always exists in a *DS-FP* schedule if the transaction set is schedulable, and if so, how to find it. We answer those questions next.

3.2 *DS-FP* Pattern Analysis

As mentioned before, we study the *DS-FP* schedulability problem in a discrete time system. In order to prove that there always exists a repeating pattern if a transaction set is schedulable by *DS-FP*, we shall first review the *Pigeonhole Principle*.

The Pigeonhole Principle [6]: If m pigeons occupy n pigeonholes and $m > n$, then at least one pigeonhole has two or more pigeons roosting in it.

In a discrete time system, a repeating pattern always exists for any successful *DS-FP* schedule because we know the fact that the execution times, validity intervals, and the number of transactions are all finite integers, and so an execution state can be defined that characterizes the progress of an execution in meeting the timing constraints for any particular time. Given infinite time, there must be a pattern repeating itself in the *DS-FP* schedule inasmuch as the number of distinct execution states is finite. The following theorem states that there must exist a fixed pattern repeat-

ing itself in a bounded time interval in the *DS-FP* schedule.

Theorem 3.1: Given an update transaction set \mathcal{T} with known C_i and \mathcal{V}_i ($1 \leq i \leq m$), if it can be scheduled by *DS-FP*, then the *DS-FP* schedule has a *fixed repeating pattern* that occurs at least once in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \Pi_{i=1}^m (\mathcal{V}_i - C_i + 1) - 1]$.

Proof. The theorem can be proved by the following two claims using induction:

1. There is a *fixed pattern* repeating itself for τ_1 in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \mathcal{V}_1 - C_1]$.
2. For any k , $1 \leq k < m$, if there is a *fixed pattern* repeating itself for the schedule of τ_1, \dots, τ_k in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \Pi_{i=1}^k (\mathcal{V}_i - C_i + 1) - 1]$, then there is a *fixed pattern* repeating itself for $\tau_1, \dots, \tau_k, \tau_{k+1}$ in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \Pi_{i=1}^{k+1} (\mathcal{V}_i - C_i + 1) - 1]$.

The first claim is obvious because there is a fixed pattern of length $(\mathcal{V}_1 - C_1)$ repeating itself from time 0. As a matter of fact, any schedule of length $(\mathcal{V}_1 - C_1)$ is a fixed pattern, thus the schedule of length $(\mathcal{V}_1 - C_1)$ from time $\mathcal{V}_m - C_m$ must repeat itself. Note that the theorem does not require that the first instance of the fixed pattern in the interval $[\mathcal{V}_m - C_m, \mathcal{V}_m - C_m + \Pi_{i=1}^m (\mathcal{V}_i - C_i + 1) - 1]$ starts exactly on time $\mathcal{V}_m - C_m$.

Now let us prove the second claim. We shall rely on the *Pigeonhole* Principle to identify two time points in two recurring patterns for transactions τ_1, \dots, τ_k such that the following two conditions are satisfied: 1) the two time points are τ_{k+1} 's release times; 2) the two time points have the same offsets within their patterns. If such two time points are identified, then the schedule of $\tau_1, \dots, \tau_k, \tau_{k+1}$ between those two time points is a fixed pattern repeating itself thereafter. This is because the schedules after those two time points are identical for transactions τ_1, \dots, τ_k , thus it is also identical for τ_{k+1} .

Suppose that the fixed pattern for τ_1, \dots, τ_k starting from time t has length L . We have $t \geq (\mathcal{V}_m - C_m)$ and $L \leq \Pi_{i=1}^k (\mathcal{V}_i - C_i + 1) - 1$. There are two cases, $L \geq (\mathcal{V}_{k+1} - C_{k+1})$ and $L < (\mathcal{V}_{k+1} - C_{k+1})$.

Case I: Suppose $L \geq (\mathcal{V}_{k+1} - C_{k+1})$. In every recurring instance of the fixed pattern of length L starting from time t , there is at least one job of τ_{k+1} since $L \geq (\mathcal{V}_{k+1} - C_{k+1})$. Let us examine the last τ_{k+1} job in each recurring instance of the pattern. Denote d to be the distance from its release time to the end of the pattern. The length of d cannot exceed $(\mathcal{V}_{k+1} - C_{k+1})$, otherwise there must be another job afterwards in the pattern in order to satisfy τ_{k+1} 's validity constraint. Since $d > 0$, it can be one of $(\mathcal{V}_{k+1} - C_{k+1})$ possible values (pigeonholes). Let us look at τ_{k+1} 's last job (pigeon) in each of the $(\mathcal{V}_{k+1} - C_{k+1} + 1)$ recurring patterns since time t . It follows from the *Pigeonhole* Principle

that there must be two jobs' release time at the same offset within their corresponding pattern instances. Denote t_1 and t_2 to be the two job release times, and $t_1 < t_2$. We then have a fixed pattern in the interval $[t_1, t_2]$ repeating itself thereafter for transactions $\tau_1, \dots, \tau_k, \tau_{k+1}$. Since $t_1 \geq t \geq (\mathcal{V}_m - C_m)$ and $t_2 < (\mathcal{V}_m - C_m) + L(\mathcal{V}_{k+1} - C_{k+1} + 1) < (\mathcal{V}_m - C_m) + \Pi_{i=1}^{k+1} (\mathcal{V}_i - C_i + 1) - 1$, we have proved the first case.

Case II: Suppose $L < (\mathcal{V}_{k+1} - C_{k+1})$. Denote $J_{k+1,w}$ to be the first τ_{k+1} 's job that executes after time t . $J_{k+1,w}$ and all its subsequent jobs appear in different instances of the pattern. There are only L possible offsets (pigeonholes) within a pattern for τ_{k+1} 's jobs to start. Let us look at the first $(L + 1)$ jobs (pigeon) of τ_{k+1} , i.e., $J_{k+1,w}$ through $J_{k+1,w+L}$. It follows from the *Pigeonhole* Principle that there must be two jobs starting at the same offset within their corresponding pattern instances. Denote t_1 and t_2 to be the two job release times in *DS-FP*, and $t_1 < t_2$. We then have a fixed pattern in the interval $[t_1, t_2]$ repeating itself for transactions $\tau_1, \dots, \tau_k, \tau_{k+1}$. Since $t_1 \geq t \geq (\mathcal{V}_m - C_m)$ and $t_2 < (\mathcal{V}_m - C_m) + (L + 1)(\mathcal{V}_{k+1} - C_{k+1}) < (\mathcal{V}_m - C_m) + \Pi_{i=1}^{k+1} (\mathcal{V}_i - C_i + 1) - 1$, we have proved the second case.

Based on the above two claims, the theorem is proved. \square

According to the theorem, if a transaction set can be scheduled by *DS-FP* in the interval $[0, (\mathcal{V}_m - C_m) + \Pi_{i=1}^m (\mathcal{V}_i - C_i + 1) - 1]$, then it is schedulable by *DS-FP* because a fixed pattern appearing in the interval repeats itself forever. Thus we have the following corollary.

Corollary 3.1: An update transaction set \mathcal{T} can be scheduled by *DS-FP* if and only if it can be scheduled by *DS-FP* in the interval $[0, (\mathcal{V}_m - C_m) + \Pi_{i=1}^m (\mathcal{V}_i - C_i + 1) - 1]$.

Although the length of the interval in Corollary 3.1 is $O(\Pi_{i=1}^m \mathcal{V}_i)$, Corollary 3.1 provides a foundation for the schedulability test of *DS-FP*.

3.3 *DS-FP* Pattern Search Algorithm

Theorem 3.1 proves the existence of a repeating pattern for a given *DS-FP* schedule. This subsection studies the properties of the pattern and provides a search algorithm that finds the shortest and earliest pattern.

Denote tuple $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$ to be the *DS-FP* schedule of length \mathcal{P}_l starting from time \mathcal{P}_s . If a *DS-FP* schedule repeats \mathcal{P} forever after time $\mathcal{P}_s + \mathcal{P}_l$, then \mathcal{P} is a *fixed pattern* of the *DS-FP* schedule. Given a pattern \mathcal{P} , *DS-FP* may not necessarily be idle immediately before \mathcal{P}_s .

Denote tuple $\mathcal{S}_\tau(t) = (d, e)$ to be the state of transaction τ at time t , where d is the distance to τ 's last job release time before time t , and e is the remaining outstanding execution time of τ at time t . $e = 0$ if τ 's last job before t is already finished. Let $\mathcal{S}_\mathcal{T}(t)$ be the combined $\mathcal{S}_\tau(t)$ states of all the transactions in the set \mathcal{T} at time t . Note once $\mathcal{S}_\mathcal{T}(t)$ is known, the *DS-FP* schedule from t onward is determined. Given a repeating pattern $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$, the fol-

lowing corollary follows directly from the fact that all transactions have the same states at times $\mathcal{P}_s + t$ and $\mathcal{P}_s + t + \mathcal{P}_l$ for $t > 0$, i.e., $\mathcal{S}_{\mathcal{T}}(\mathcal{P}_s + t) = \mathcal{S}_{\mathcal{T}}(\mathcal{P}_s + t + \mathcal{P}_l)$.

Corollary 3.2: If $\mathcal{P} = (\mathcal{P}_s, \mathcal{P}_l)$ is a pattern repeating itself from time \mathcal{P}_s , then $(\mathcal{P}_s + t, \mathcal{P}_l)$ ($t > 0$) is also a pattern repeating itself from time $\mathcal{P}_s + t$.

We now prove the next lemma.

Lemma 3.1: Given all the patterns $\mathcal{P}, \mathcal{P}', \dots$ of a *DS-FP* schedule for transaction set \mathcal{T} , let \mathcal{P} be a pattern with the minimum length among all patterns, i.e., $\mathcal{P}_l \leq \mathcal{P}'_l$ for any other pattern \mathcal{P}'_l . Then \mathcal{P}'_l is a multiple of \mathcal{P}_l , i.e., $\mathcal{P}'_l = N\mathcal{P}_l$ where N is a positive integer.

Proof. Let $t_1 = \mathcal{P}'_s + \mathcal{P}'_l * n_1$ ($n_1 > 0$ is an integer) such that $t_1 > \mathcal{P}_s$. Both (t_1, \mathcal{P}_l) and (t_1, \mathcal{P}'_l) are patterns.

We prove the lemma by contradiction. Suppose \mathcal{P}'_l is not a multiple of \mathcal{P}_l and $\mathcal{P}'_l = \mathcal{P}_l * r - s$, $r \geq 2$, and $0 < s < \mathcal{P}_l$. We have the state $\mathcal{S}_{\mathcal{T}}((t_1 + \mathcal{P}'_l) + s) = \mathcal{S}_{\mathcal{T}}(t_1 + \mathcal{P}_l * r) = \mathcal{S}_{\mathcal{T}}(t_1) = \mathcal{S}_{\mathcal{T}}(t_1 + \mathcal{P}'_l)$. It follows that the pattern $(t_1 + \mathcal{P}'_l, s)$ repeats itself from time $(t_1 + \mathcal{P}'_l)$ with length s . As $0 < s < \mathcal{P}_l$, this contradicts the fact that \mathcal{P}_l is the minimum length among all repeating patterns. So \mathcal{P}'_l must be a multiple of \mathcal{P}_l . \square

In the proof above, since \mathcal{P}'_l is a multiple of \mathcal{P}_l , $(\mathcal{P}'_s, \mathcal{P}_l)$ must also be a pattern.

Corollary 3.3: Given all the patterns of a *DS-FP* schedule, let \mathcal{P} be a pattern with the minimum \mathcal{P}_l . For any other pattern \mathcal{P}' , $(\mathcal{P}'_s, \mathcal{P}_l)$ is also a repeating pattern.

Lemma 3.1 and Corollary 3.3 imply that there exists a shortest pattern \mathcal{P} that is also the earliest. Any other pattern \mathcal{P}' could be derived from \mathcal{P} . \mathcal{P}' could be of the same length but with some offset from a \mathcal{P} 's repeat; \mathcal{P}' could be a multiple of \mathcal{P} 's repeats; or \mathcal{P}' could be a multiple of \mathcal{P} 's repeats with some offset.

Lemma 3.2: If \mathcal{P}' and \mathcal{P}'' are two different repeating patterns of a *DS-FP* schedule, then $(\mathcal{P}'_s, \mathcal{P}''_l)$ and $(\mathcal{P}''_s, \mathcal{P}'_l)$ are also repeating patterns.

Proof. Let \mathcal{P} be the shortest and earliest pattern. According to Lemma 3.1, \mathcal{P}''_l is a multiple of \mathcal{P}_l . According to Corollary 3.3, $(\mathcal{P}'_s, \mathcal{P}_l)$ is also a repeating pattern. Because $(\mathcal{P}'_s, \mathcal{P}_l)$ is a pattern and \mathcal{P}''_l is a multiple of \mathcal{P}_l , $(\mathcal{P}'_s, \mathcal{P}''_l)$ is also a repeating pattern.

By the same argument, $(\mathcal{P}''_s, \mathcal{P}'_l)$ is a repeating pattern. \square

Given transaction set \mathcal{T} of size m , we call \mathcal{P}^i a pattern of the first i ($1 \leq i \leq m$) highest priority transactions (τ_1, \dots, τ_i) by ignoring all other lower priority transactions $\tau_{i+1}, \dots, \tau_m$ in the schedule. In other words, \mathcal{P}^i is a pattern of the transaction set consisting of only the first i highest priority transactions.

Lemma 3.3: If \mathcal{P}^i is the shortest and earliest pattern of the first i ($1 \leq i < m$) highest priority transactions, and \mathcal{P}^{i+1} is the shortest and earliest pattern of the first $i + 1$ highest priority transactions, then

1. $\mathcal{P}_s^i \leq \mathcal{P}_s^{i+1}$.
2. \mathcal{P}_l^{i+1} is a multiple of \mathcal{P}_l^i .

Proof. By ignoring the schedule of τ_{i+1} in \mathcal{P}^{i+1} , \mathcal{P}^{i+1} is also a repeating pattern of the first i highest priority transactions. By definition, $\mathcal{P}_s^i \leq \mathcal{P}_s^{i+1}$. By Lemma 3.1, \mathcal{P}_l^{i+1} is a multiple of \mathcal{P}_l^i . \square

We now present the pattern search algorithm. The algorithm follows the idea in the proof of Theorem 3.1. It searches for the pattern of the first i ($2 \leq i \leq m$) highest priority transactions based on the repeating pattern of the first $i - 1$ highest priority transactions. After the algorithm completes for the lowest priority transaction, it returns the pattern for the transaction set.

Algorithm 3.1 SearchPattern:

Input: A successful *DS-FP* schedule.

Output: The earliest and shortest pattern \mathcal{P}^m .

```

1  $\mathcal{P}^1 \leftarrow (0, \mathcal{V}_1 - C_1)$ ; // Pattern of the first transaction.
2 for  $i = 2$  to  $m$  do
3   // Find the pattern when adding the next transaction.
4    $\mathcal{P}^i \leftarrow \text{SearchNextTask}(i, \mathcal{P}^{i-1})$ ;
5 od
6 return  $\mathcal{P}^m$ ;

```

Alg. 3.1 invokes Alg. 3.2 whose input is the pattern of the first $i - 1$ ($1 < i \leq m$) highest priority transactions, and output is the pattern of the first i highest priority transactions. Alg. 3.2 scans the *DS-FP* schedule for the jobs of the i^{th} highest priority transaction to find the first two jobs such that each starts at the same offset within its corresponding input pattern \mathcal{P}^{i-1} . The schedule between these two release times forms the output pattern for the first i highest priority transactions.

Algorithm 3.2 SearchNextTask:

Input: Pattern \mathcal{P}^{i-1} of transactions $\tau_1, \dots, \tau_{i-1}$.

Output: Pattern \mathcal{P}^i of transactions $\tau_1, \dots, \tau_{i-1}, \tau_i$.

```

1  $k \leftarrow 1$ ;
2  $r_{i,j} \leftarrow$  first  $\tau_i$  release time after  $\mathcal{P}_s^{i-1}$ ;
3  $maxL \leftarrow 1 +$  No. of idle slots in  $\mathcal{P}^{i-1}$ ;
4 while ( $k \leq maxL$ ) do
5   for  $r = r_{i,j}$  to  $r_{i,j+k-1}$  do
6     if  $((r_{i,j+k} - r) \% \mathcal{P}_l^{i-1} = 0)$ 
7       then
8          $\mathcal{P}^i \leftarrow (r, r_{i,j+k} - r)$ ; //Find the shortest pattern.
9         // The next loop finds the earliest pattern.
10      while  $(\mathcal{S}_{\mathcal{T}}(\mathcal{P}_s^i - 1) = \mathcal{S}_{\mathcal{T}}(\mathcal{P}_s^i - 1 + \mathcal{P}_l^i))$  do

```

```

11      $\mathcal{P}^i \leftarrow (\mathcal{P}_s^i - 1, \mathcal{P}_l^i);$ 
12     od
13     return  $\mathcal{P}^i;$ 
14 fi
15 od
16      $k \leftarrow k + 1;$ 
17 od
18 return No pattern found;

```

Note that in Alg. 3.2, τ_i can only be scheduled in the idle slots of the input pattern \mathcal{P}^{i-1} . According to the *Pigeonhole Principle*, Alg. 3.2 does not need to examine more jobs than the number of idle slots plus 1 in \mathcal{P}^{i-1} . In other words, the while loop of line 4 in Alg. 3.2 does not need to loop more than the number of idle slots in \mathcal{P}^{i-1} plus 1. Thus the condition at line 6 can be true at least once before the while loop beginning from line 4 ends.

Line 8 in Alg. 3.2 produces the shortest pattern starting from the release time of one of τ_i 's jobs. Given a job $\tau_{i,j+k}$ that satisfies condition $((r_{i,j+k} - r) \% \mathcal{P}_l^{i-1} = 0)$ at line 6, the while loop at line 10 cannot not run for more than $\mathcal{V}_i - C_i$ times, otherwise the algorithm must have returned \mathcal{P}^i based on $\tau_{i,j+k}$'s previous job because $((r_{i,j+k-1} - r) \% \mathcal{P}_l^{i-1} = 0)$ implies that the pattern $(r, r_{i,j+k-1} - r)$ can be returned. This while loop shifts the identified pattern to its earliest possible starting time. Also note that the while loop cannot move back to τ_i 's first job $J_{i,0}$ ($i \geq 2$) because the release time of $J_{i,0}$ (i.e., time 0) is not the beginning time of $J_{i,0}$'s execution in the *DS-FP* algorithm. As an example, in Figure. 2 (b), $\mathcal{P} = (7, 12)$ is the shortest pattern. At time 6, $\mathcal{S}_{\tau_2}(6) = (6, 1)$ and $\mathcal{S}_{\tau_2}(18) = (4, 1)$. This implies $\mathcal{S}_{\mathcal{T}}(\mathcal{P}_s - 1) \neq \mathcal{S}_{\mathcal{T}}(\mathcal{P}_s - 1 + \mathcal{P}_l)$ thus \mathcal{P} is the earliest pattern and can not be moved back further. Here $\mathcal{S}_{\tau_2}(6) \neq \mathcal{S}_{\tau_2}(18)$ because the release time of $J_{2,0}$ (time 0) is not the beginning time of $J_{2,0}$'s execution.

Theorem 3.2: \mathcal{P}^m returned by Alg. 3.1 is the shortest and earliest pattern if such a pattern exists.

Proof. We shall prove that if the input to Alg. 3.2 is the shortest and earliest pattern, so is the output.

We first prove that Alg. 3.2 returns a pattern. Alg. 3.2 returns only when the condition at line 6 is true. The condition implies that r and $r_{i,j+k}$ are of the same offsets within their respective input patterns. So \mathcal{P}^i derived at line 8 is a pattern for transactions $\tau_1, \dots, \tau_{i-1}, \tau_i$. Furthermore, the condition of line 10 guarantees that \mathcal{P}^i remains to be a pattern when it is shifted along the timeline.

We then prove that the returned \mathcal{P}^i is the shortest. Let us examine \mathcal{P}^i produced at line 8. Assume that the shortest pattern is of length L and $L \leq \mathcal{P}_l^i$, then according to Corollary 3.3 (\mathcal{P}_s^i, L) must be a pattern. The algorithm indicates that τ_i must have a job J released at time $\mathcal{P}_s^i + L$, which is earlier than or equal to $\mathcal{P}_s^i + \mathcal{P}_l^i$. According to Lemma 3.3,

L is a multiple of \mathcal{P}_l^{i-1} . This means that J satisfies the condition at line 6. Since (\mathcal{P}_s^i, L) is the shortest, J should be the first examined job that satisfies the condition. In other words, $L = \mathcal{P}_l^i$. Finally, since \mathcal{P}^i at line 8 is the first pattern that starts with a τ_i 's release time, the while loop at line 10 guarantees that the returned \mathcal{P}^i is the earliest pattern for the first i highest priority transactions.

We have now proved that if the input to Alg. 3.2 is the shortest and earliest pattern, so is the output. We also know that line 1 in Alg. 3.1 assigns the shortest and earliest pattern for τ_1 . By induction, \mathcal{P}^m returned by Alg. 3.1 is the shortest and earliest pattern of the transaction set. \square

Alg. 3.1 runs in pseudo-polynomial time and has time complexity $O(m(\prod_{i=1}^m \mathcal{V}_i)^2)$. However, it can be improved to $O(m \prod_{i=1}^m \mathcal{V}_i)$ if an array of size $O(\mathcal{P}_l^{i-1})$ can be used to keep relative offsets of τ_i 's jobs within their corresponding patterns (i.e., \mathcal{P}^{i-1}). Note that the while loop at line 10 is only executed once although it is within the two outer loops. It loops at most $\mathcal{V}_i - C_i$ times. Thus it is ignored in the complexity calculation.

3.4 DS-FP Schedulability Test Algorithm

Alg. 3.1 also implies a schedulability test algorithm. We begin the schedulability test with τ_1 . Given a subset of transactions $\tau_1, \dots, \tau_{i-1}$ ($1 < i \leq m$) that has been tested, we test transaction τ_i by adding the transaction to the subset until an added transaction is not schedulable or a pattern for all transactions is found. Given transaction τ_i , we schedule it along with the schedule of the higher priority transactions $\tau_1, \dots, \tau_{i-1}$, for which a pattern has already been found. Alg. 3.3 and Alg. 3.4 are modified versions of Alg. 3.1 and Alg. 3.2 for the schedulability test, respectively.

Algorithm 3.3 Schedulability Test:

Input: A transaction set \mathcal{T} .

Output: Whether \mathcal{T} is schedulable.

```

1  $\mathcal{P}^1 \leftarrow (0, \mathcal{V}_1 - C_1);$  // Pattern of the first transaction.
2 for  $i = 2$  to  $m$  do
3     if  $(\text{TestNextTask}(i, \mathcal{P}^{i-1}) = \text{FALSE})$ 
4         then return  $\mathcal{T}$  is unschedulable; fi
5 od
6 return  $\mathcal{T}$  is schedulable;

```

Algorithm 3.4 TestNextTask:

Input: Pattern \mathcal{P}^{i-1} of transactions $\tau_1, \dots, \tau_{i-1}$.

Output: Returns TRUE and pattern \mathcal{P}^i of transactions $\tau_1, \dots, \tau_{i-1}, \tau_i$ if a pattern of those transactions exists. Otherwise, returns FALSE.

```

1 Schedule up to, including  $\tau_i$ 's first request after  $\mathcal{P}_s^{i-1}$ ;
2 if (Line 1 fails)
3     then return FALSE; fi
4  $r_{i,j} \leftarrow \tau_i$ 's first release time since  $\mathcal{P}_s^{i-1}$ ;

```

```

5  $k \leftarrow 1$ ;
6 while ( $k \leq \mathcal{P}_l^{i-1}$ ) do
7   Schedule  $r_{i,j+k}$ ;
8   if (Line 7 fails)
9     then return FALSE; fi
10  for  $r = r_{i,j}$  to  $r_{i,j+k-1}$  do
11    if  $((r_{i,j+k} - r) \% \mathcal{P}_l^{i-1} = 0)$ 
12      then // Found the shortest pattern.
13         $\mathcal{P}^i \leftarrow (r, r_{i,j+k} - r)$ ;
14        return TRUE;
15    fi
16  od
17   $k \leftarrow k + 1$ ;
18 od

```

If Alg. 3.3 returns TRUE, it also produces the shortest pattern and the *DS-FP* schedule. The following example illustrates how the algorithm works.

Example 3.3: Consider a set of three transactions $\{\tau_1, \tau_2, \tau_3\}$ with execution times 1, 1, 2, and validity intervals 3, 7, 14, respectively. It is not schedulable by *ML* because τ_3 is finished by 8, which is more than $\frac{V_3}{2} = 7$. Now we test whether it can be scheduled by *DS-FP* or not.

Figure 4 (a) corresponds to line 1 of Alg. 3.3. It shows the pattern of τ_1 .

Figure 4 (b) depicts the result of invoking Alg. 3.4 for τ_2 . There is only one idle time slot in $\{\tau_1\}$'s pattern, so the release times of two consecutive jobs $J_{2,1}$ and $J_{2,2}$ after $\mathcal{P}_s^1 = 0$ forms a pattern $\mathcal{P}^2 = (5, 6)$.

Figure 4 (c) depicts the result of invoking Alg. 3.4 for τ_3 . There are two idle time slots in $\{\tau_1, \tau_2\}$'s pattern $\mathcal{P}^2 = (5, 6)$, and the algorithm examines three consecutive jobs $J_{3,1}$, $J_{3,2}$, and $J_{3,3}$ after $\mathcal{P}_s^2 = 5$ to find an output pattern $\mathcal{P}^3 = (9, 18)$. Note that $r_{3,1}$ has an offset 4 within the pattern $\mathcal{P}^2 = (5, 6)$, while $r_{3,2}$ has an offset 2 within its corresponding pattern $\mathcal{P}^2 = (17, 6)$, and $r_{3,3}$ has an offset 4 within its corresponding pattern $\mathcal{P}^2 = (23, 6)$. The offset of $r_{3,3}$ matches that of $r_{3,1}$. So Alg. 3.4 goes to line 13, and Alg. 3.3 returns that the transaction set is schedulable.

Note that the pattern returned from Alg. 3.3 is the *shortest* but not the *earliest*. The shortest and earliest pattern for \mathcal{P}^3 is (8, 18), one time unit earlier than the starting time of \mathcal{P}^3 returned from Alg. 3.3. This earliest pattern can be returned by Alg. 3.1, which is actually calculated at lines 10 and 11 in Alg. 3.2. \square

Given a *DS-FP* schedule, there always exists a repeating pattern and our schedulability test algorithm can be applied. However, the space and time complexity of the algorithm is high. The question remains if there is a better schedulability test that is more efficient. On the other hand, a repeating pattern in the *DS-FP* schedule can be found off-line and used repeatedly to construct the *DS-FP* schedule. This can significantly reduce the on-line scheduling overhead of the *DS-FP* algorithm.

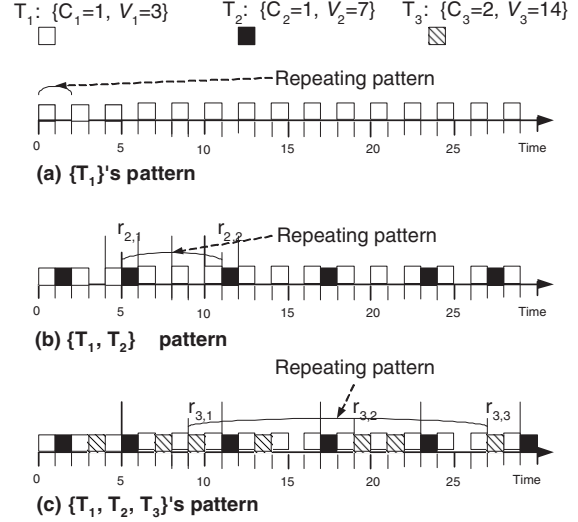


Figure 4. Illustration of the schedulability test algorithm

3.5 *DS-FP* in Continuous Time Systems

So far we assume discrete time systems for *DS-FP*. Now we move on to the schedulability discussions of *DS-FP* in continuous time systems. Given a *DS-FP* schedule, it can be proved that a repeating pattern still exists if only rational numbers are considered for transaction parameters (i.e., validity intervals and execution times). Denote l to be the least common multiple of all the denominators of all those rational numbers. If we measure time in the unit of $\frac{1}{l}$, then we again have an integer problem which has a pattern for a successful *DS-FP* schedule. This schedule is the same as the one that only has transaction parameters with original rational numbers although their granularities are different.

However, if execution times or validity intervals can be real numbers, it may not be possible to identify such a repeating pattern in a *DS-FP* schedule. We shall illustrate this with the following example.

Example 3.4: Consider a set of two transactions $\{\tau_1, \tau_2\}$ with execution times 1 and $1 + d$, and validity intervals 5 and 9 respectively. Suppose that d is an infinitesimally small real number. Figure 5 (a) depicts a schedule of the transaction set under *DS-FP*. Let i to be the largest integer such that $3 - i \times d > 1$, i.e., $i = \lfloor \frac{2}{d} \rfloor$. $r_{2,1}, r_{2,2}, \dots, r_{2,i}$ occur in every other repeating pattern of τ_1 . In addition, $\forall k, (1 \leq k \leq i)$, the offset of $r_{2,k}$ within τ_1 's pattern \mathcal{P} is $3 - k \times d$. There exists no pattern for τ_2 's first i jobs. Hence there exists no pattern from time 0 to $t = 2\mathcal{P}i = 8\lfloor \frac{2}{d} \rfloor$. Time t can be arbitrarily large if d is infinitesimally small. In other words, if execution time C_2 of τ_2 is a real number infinitesimally close to 1, there exists no repeating pattern for the *DS-FP* schedule. Note that the transaction set has fi-

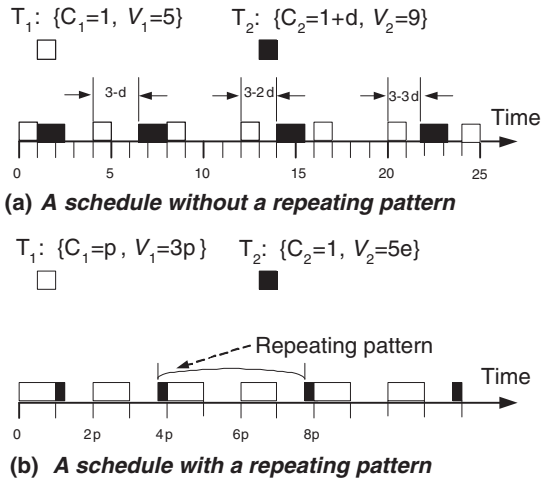


Figure 5. *DS-FP* schedules for transaction sets with real number parameters

nite number of transactions, and finite values for execution times and validity intervals.

We can also prove that the transaction set is schedulable by *DS-FP* using induction. We know $J_{2,0}$ and $J_{2,1}$ are schedulable. We can easily prove that if $J_{2,i}, i > 0$ is schedulable, so is $J_{2,i+1}$. Another proof follows from Theorem 2.1 because the transaction set is obviously schedulable by *ML*. \square

Our observation from Example 3.4 is the following: given an arbitrarily large time t ($t \rightarrow +\infty$), there always exists a transaction set with finite number of transactions and finite real number parameters that has a successful *DS-FP* schedule without any repeating pattern in the interval $[0, t]$. However, there also exist transaction sets with finite number of transactions and finite real number parameters that have successful *DS-FP* schedules with repeating patterns, which is illustrated by the following example.

Example 3.5: Define two real numbers $p = \pi = 3.14159\dots$ and $e = 2.71828\dots$. Consider a set of two transactions $\{\tau_1, \tau_2\}$ with execution times $p, 1$, and validity intervals $3p, 5e$, respectively. Figure 5 (b) depicts the *DS-FP* schedule of the transaction set with a repeating pattern. \square

4 Conclusions and Future Work

Distinct from past studies of maintaining the freshness (or temporal validity) of real-time data that adopt the periodic task model, *DS-FP* adopts the *aperiodic* task model. In *DS-FP*, the deadlines of jobs and the separation of two consecutive jobs of an update transaction are adjusted judiciously so that the farthest distance of the sampling times of two consecutive jobs can be achieved. However, the

aperiodic model in *DS-FP* poses a great challenge for its schedulability test. This paper addresses this issue in discrete time system by providing a necessary and sufficient schedulability condition for *DS-FP*. We prove the existence of a repeating pattern for any successful *DS-FP* schedule, and derive the corresponding schedulability test algorithm. While this provides a schedulability analysis of *DS-FP* in discrete time systems, there are still unanswered questions for *DS-FP* schedulability in continuous time systems. Furthermore, the achievable time complexity of our schedulability test algorithm is $O(m \prod_{i=1}^m V_i)$, thus it is highly desirable to improve the efficiency of the algorithm.

References

- [1] S. K. Baruah, A. K. Mok, L. E. Rosier, "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor," *IEEE Real-Time Systems Symposium*, December 1990.
- [2] A. Burns and R. Davis, "Choosing task periods to minimise system utilisation in time triggered systems," in *Information Processing Letters*, 58 (1996), pp. 223-229.
- [3] D. Chen and A. K. Mok, "Scheduling Similarity-Constrained Real-Time Tasks," pp. 215-221, *ESA/VLSI*, 2004.
- [4] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, pp. 1261-1269, October 1989.
- [5] R. Gerber, S. Hong and M. Saksena, "Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes," *IEEE Real-Time Systems Symposium*, December 1994.
- [6] R. P. Grimaldi, "Discrete and Combinatorial Mathematics: An Applied Introduction," *Addison-Wesley*, 4th Edition, pp. 244-248, 1998.
- [7] T. Gustafsson, J. Hansson, "Data Management in Real-Time Systems: a Case of On-Demand Updates in Vehicle Control Systems," *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 182-191, 2004.
- [8] T. Gustafsson and J. Hansson, "Dynamic on-demand updating of data in real-time database systems," *ACM SAC*, 2004.
- [9] C. C. Han, K. J. Lin and J. W.-S. Liu, "Scheduling Jobs with Temporal Distance Constraints," *Siam Journal of Computing*, Vol. 24, No. 5, pp. 1104 - 1121, October 1995.
- [10] K. D. Kang, S. Son, J. A. Stankovic, and T. Abdelzaher, "A QoS-Sensitive Approach for Timeliness and Freshness Guarantees in Real-Time Databases," *EuroMicro Real-Time Systems Conference*, June 2002.
- [11] T. Kuo and A. K. Mok, "Real-Time Data Semantics and Similarity-Based Concurrency Control," *IEEE Real-Time Systems Symposium*, December 1992.
- [12] T. Kuo and A. K. Mok, "SSP: a Semantics-Based Protocol for Real-Time Data Access," *IEEE Real-Time Systems Symposium*, December 1993.
- [13] S. Ho, T. Kuo, and A. K. Mok, "Similarity-Based Load Adjustment for Real-Time Data-Intensive Applications," *IEEE Real-Time Systems Symposium*, 1997.
- [14] K.Y. Lam, M. Xiong, B. Liang and Y. Guo, "Statistical Quality of Service Guarantee for Temporal Consistency of Real-time Data Objects," *IEEE Real-Time Systems Symposium*, 2004.
- [15] J. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks," *Performance Evaluation*, 2(1982), 237-250.
- [16] C. L. Liu, and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, 20(1), 1973.

- [17] D. Locke, "Real-Time Databases: Real-World Requirements," in *Real-Time Database Systems: Issues and Applications*, edited by Azer Bestavros, Kwei-Jay Lin and Sang H. Son, Kluwer Academic Publishers, pp. 83-91, 1997.
- [18] K. Ramamritham, "Real-Time Databases," *Distributed and Parallel Databases* 1(1993), pp. 199-226, 1993.
- [19] 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.
- [20] 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.
- [21] J. A. Stankovic, S. Son, and J. Hansson, "Misconceptions About Real-Time Databases," *IEEE Computer*, Vol. 32, No. 6, pp. 29-36, June 1999.
- [22] M. Xiong, K. Ramamritham, J. A. Stankovic, D. Towsley, and R. M. Sivasankaran, "Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics," *IEEE Transactions on Knowledge and Data Engineering*, 14(5), 1155-1166, 2002.
- [23] M. Xiong and K. Ramamritham, "Deriving Deadlines and Periods for Real-Time Update Transactions," *IEEE Transactions on Computers*, 53(5), pp. 567-583, 2004.
- [24] M. Xiong, S. Han, and K.Y. Lam, "A Deferrable Scheduling Algorithm for Real-Time Transactions Maintaining Data Freshness," *IEEE Real-Time Systems Symposium*, 2005.
- [25] M. Xiong, S. Han, K. Y. Lam and D. Chen. "Deferrable Scheduling for Maintaining Real-Time Data Freshness: Algorithms, Analysis and Results," *IEEE Transactions on Computers*, to appear. A longer version is available as Technical Report TR-07-44, Dept. of Computer Sciences, Univ. of Texas at Austin, Sept. 2007.