On Earliest Deadline First Scheduling for Temporal Consistency Maintenance

Ming Xiong Qiong Wang
Bell Laboratories, Alcatel-Lucent
{xiong, qwang}@research.bell-labs.com

Krithi Ramamritham
India Institute of Technology Bombay
krithi@cse.iitb.ac.in

Abstract

A real-time object is one whose state may become invalid with the passage of time. A temporal validity interval is associated with the object state, and the real-time object is temporally consistent *if its temporal validity interval has not expired. Clearly, the problem of maintaining temporal consistency of data is motivated by the need for a real-time system to track its environment correctly. Hence, sensor transactions must be able to execute periodically and also each instance of a transaction should perform the relevant data update before its deadline.*

Unfortunately, the period and deadline assignment problem for periodic sensor transactions has not received the attention that it deserves. An exception is the More-Less scheme, which uses the Deadline Monotonic *(DM) algorithm for scheduling periodic sensor transactions. However, there is no work addressing this problem from the perspective of dynamic priority scheduling. In this paper, we examine the problem of temporal consistency maintenance using the* Earliest Deadline First *(*EDF*) algorithm in three steps:*

First, the problem is transformed to another problem with a sufficient (but not necessary) condition for feasibly assigning periods and deadlines. An optimal solution for the problem can be found in linear time*, and the resulting processor utilization is characterized and compared to a traditional approach. Second, an algorithm to search for the optimal periods and deadlines is proposed. The problem can be solved for sensor transactions that require any arbitrary deadlines. However, the optimal algorithm does not scale well when the problem size increases. Hence, thirdly, we propose a heuristic search-based algorithm that is more efficient than the optimal algorithm and is capable of finding a solution if one exists.*

1 Introduction

A real-time data object, e.g., the speed or position of a vehicle, or the temperature in an engine, is *temporally consistent* (also known as *temporally valid*) if its value reflects the current status of the corresponding entity in the environment. This

is usually achieved by associating the value with *a temporal validity interval* [15, 19, 11, 23, 22, 25, 24]. For example, if position or velocity changes that can occur in 10 seconds do not affect any decisions made about the navigation of a vehicle, then the temporal validity interval associated with these data elements is at least 10 seconds.

One important design goal of real-time and embedded database systems is to always keep the real-time data temporally consistent. Otherwise, the systems cannot detect and respond to environmental changes in a timely fashion. Thus, sensor transactions that sample the latest status of the entities need to periodically refresh the values of real-time data objects before their old values expire. Given *temporal consistency* requirements for a set of sensor transactions, the problem of designing such sensor transactions encompasses two issues [22]: (1) the determination of the update period and deadline for each transaction from their *temporal consistency* requirements; and (2) the schedulability of periodic sensor transactions. Minimizing the update workload for maintaining temporal consistency is an important design issue because: (1) it allows a system to accomodate more sensor transactions; (2) it allows a real-time embedded system to consume less energy; and (3) it leaves more processor capacity for other workload (e.g., transactions triggered due to detected environmental changes).

Temporal consistency maintenance can be described as a real-time scheduling problem: Given m transactions (or tasks) with computation time (C_i) and validity interval constraint ($\mathcal{V}_i$), determine periodic tasks with deadline (D_i) and period (P_i) of minimum CPU utilization such that:

1. $D_i + P_i \leq \mathcal{V}_i$, and

2. The task system is schedulable by a scheduling algorithm α.

A traditional method for maintaining temporal consistency is the Half-Half (*HH*) scheme [15, 11] in which the update period and deadline for a real-time data object is set to be half of its temporal validity interval. To further reduce the update workload, the More-Less (*ML*) scheme is proposed [3, 22]. Deadline monotonic (*DM*), a fixed priority scheduling algorithm, is used in *ML* [3, 22, 25] to maintain temporal consistency. *ML* significantly reduces the update workload compared to *HH*. In [22], *ML* is designed only for those cases in which the assigned deadline of a sensor transaction is not greater than its corresponding period. In [3], a *DM* based approach is proposed to allow transactions with deadlines greater than their periods. If *arbitrary* deadlines, i.e., deadlines that can be less than, equal to, or greater than their corresponding periods, are allowed, then more sensor transactions with derived periods and deadlines can be scheduled by the system and feasibility of sensor transactions can be improved. However, there is no work addressing the deadline and period assignment problem from the perspective of dynamic priority scheduling. This paper presents our recent studies on this topic.

We take the first step towards designing and analyzing approaches using *EDF* scheduling for temporal consistency maintenance, and shed light on the performance of those various approaches. The problem of using *EDF* scheduling for temporal consistency maintenance is first transformed to another problem with a sufficient (but not necessary) feasibility condition by having a deadline no greater than its period. An optimal solution for the problem can be found in *linear time*, and its

utilization is characterized and compared to a traditional approach. Second, a *branch and bound* based optimal search algorithm is proposed for the more general problem. The problem can be solved for sensor transactions that require any arbitrary deadlines. However, the optimal algorithm does not scale well when the problem size increases. Hence, thirdly, we propose a *branch and bound* based heuristic search algorithm that is more efficient than the optimal algorithm and is capable of finding a solution if one exists.

This paper is organized as follows: Section 2 reviews related work. Section 3 introduces the concept of temporal consistency and prior solutions for temporal consistency maintenance based on fixed priority scheduling algorithms. Section 4 gives a detailed analysis of designing *ML* using *EDF* scheduler with a sufficient feasibility condition when $D_i \leq P_i$ holds. Section 5 presents two search algorithms, namely $\mathcal{OS}_{EDF}$ and $\mathcal{HS}_{EDF}$, for finding optimal and heuristic solutions for the problem, respectively when *arbitrary* deadlines are allowed. This section also shows that $\mathcal{HS}_{EDF}$ can produce good solutions efficiently. Finally, Section 7 concludes the paper.

2 Related Work

There has been extensive work in RTDBSs for guaranteeing the *validity constraint* (or *similarity-constraint*) of real-time data [19, 9, 10, 11, 4, 23, 8, 25, 6, 22, 24]. RTDB systems must maintain data temporal consistency in addition to logical consistency. Gustafsson and Hansson [6] present a vehicular application with embedded engine control systems, and an on-demand scheduling algorithm for enforcing base and derived data freshness. They also propose an algorithm (ODTB) [5] for updating data items that can skip unnecessary updates allowing for better utilization of the CPU in the vehicular application. In the model introduced in [19], 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 are greater than the absolute thresholds, or age differences among objects are greater than the 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 [19]. In [9, 10], real-time data semantics are investigated and a class of real-time data access protocols called SSP (Similarity Stack Protocols) is proposed. The correctness of SSP is based on the concept of similarity which allows different but sufficiently timely data to be used in a computation without adversely affecting the outcome. In [11], similarity-based principles are coupled with the *Half-Half* approach to adjust real-time transaction load by skipping the execution of task instances. The concept of *data-deadline* is proposed in [23]. It also proposes data-deadline based scheduling, forced-wait and similarity-based scheduling techniques to maintain the temporal validity of real-time data and meet transaction deadlines in RTDBSs. The aforementioned work assumes that sensor transactions are executed periodically, and their deadlines and periods are given. As a result, it provides no answer to the period and deadline assignment problem for maintaining temporal consistency.

The *More-Less* scheme [22], which will be reviewed in Section 3, solves the period and deadline assignment problem for

maintaining temporal consistency with *Deadline Monotonic* scheduling, a *fixed* priority scheduling algorithm. On the other hand, the deferrable scheduling approach [24] is a *fixed* priority scheduling algorithm that follows an aperiodic execution model. It derives relative deadlines and irregular separations for consecutive jobs adaptively to maintain temporal validity. The scheduling overhead is much higher than the periodic scheduling approaches. It also lacks the support theory to schedule transactions with arbitrary deadlines. In this paper, we focus on periodic approaches. Distinct from past work based on *fixed* priority scheduling, we present our novel approaches based on *dynamic* priority scheduling.

Note that in this paper we focus on reducing update workload, and more importantly, on improving feasibility of sensor transactions using *EDF* scheduling. This is orthogonal to the ideas of using data deadlines, forced wait [23] and data similarity [11, 23] to improve real-time transaction performance or reduce update workload. However, our design approach can be coupled with those ideas to handle sensor and triggered transactions. Our main contributions are to propose optimal and efficient algorithms for temporal consistency maintenance using *EDF* scheduling for *arbitrary* deadlines, and their corresponding analysis.

In many computer-controlled real-time applications, the application tasks often have a maximal acceptable latency while small latency is preferred. The interaction between choosing task periods to meet the individual latency requirements and scheduling the resulting task set is investigated in [17] using earliest deadline first scheduling. Similar problems are also investigated in [18] in which Seto et al. present algorithms based on rate monotonic scheduling to determine optimal periods for each task in a task set. Our work is different from [17, 18] as we derive periods and deadlines based on temporal consistency requirements, which are not considered in [17, 18].

3 Background

In this section, we introduce the concept of *temporal consistency*, and present prior work for maintaining temporal consistency of real-time objects based on fixed priority scheduling algorithms.

3.1 Temporal Consistency: Definition and Maintenance

To monitor the states of objects faithfully, *a real-time object must be refreshed by a sensor transaction before it becomes invalid (i.e., its temporal validity interval expires)*. The actual length of the temporal validity interval of a real-time object is application dependent. Temporal consistency (a.k.a. absolute consistency [15]) is proposed to determine the temporal validity of a real-time data object based on its update time by assuming that the highest rate of change in a real-time data object is known. Sensor transactions have periodic jobs, which are generated by intelligent sensors that sample the value of real-time objects. One important design goal of RTDBs is to guarantee that real-time data remain fresh, i.e., they are always valid. We assume that a sensor always samples the value of a real-time data X_i at $r_{i,j}$ $(j = 1, 2...)$, the beginning of its j^{th}

Symbol	Definition
X_i	Real-time data i
τ_i	Sensor transaction updating X_i
$J_{i,j}$	The jth job of τ_i
C_i	Computation time of 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 τ_i
D_i	Relative deadline of τ_i
$\mathcal{U}$	Processor workload of $\{\tau_i\}_{i=1}^{m}$

Table 1. Symbols and definitions.

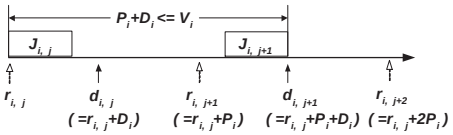

Figure 1. Illustration of *More-Less*

i	C_i	$\mathcal{V}_i$	*More-Less*	
			D_i	P_i
1	2	10	2	8
2	5	30	7	23
3	9	37	20	17

Table 2. Example 3.1

update period. It should be clear that $r_{i,j} = (j-1) \cdot P_i$ $(j = 1, 2...)$, where P_i is the period for updating X_i. A data value for real-time data X_i sampled at time $r_{i,j}$ will be valid for $\mathcal{V}_i$ following that $r_{i,j}$ up to $(r_{i,j} + \mathcal{V}_i)$.

Definition 3.1: A real-time data object (X_i) at time t is temporally consistent (or temporally valid) if, for its update job $J_{i,j}$ finishing last before time 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., $r_{i,j} + \mathcal{V}_i \geq t$. □

From here on, $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ refers to a set of periodic sensor transactions and $\mathcal{X} = \{X_i\}_{i=1}^{m}$ refers to a set of real-time data. All real-time data are kept in main memory. Associated with X_i $(1 \leq i \leq m)$ is a validity interval of length $\mathcal{V}_i$: transaction τ_i $(1 \leq i \leq m)$ updates the corresponding data X_i with validity length $\mathcal{V}_i$. Because each sensor transaction updates different data, no concurrency control is considered for sensor transactions. We assume that the time of the system is discrete, and the system is *synchronous*, i.e., all the first jobs of sensor transactions are initiated at the same time. 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. The relative deadline D_i of the j^{th} job $J_{i,j}$ of sensor transaction τ_i is defined as D_i = $d_{i,j}$ - $r_{i,j}$, where $d_{i,j}$ is the absolute deadline of $J_{i,j}$, and $r_{i,j}$ is the sampling (or release) time of $J_{i,j}$. Formal definitions of the frequently used symbols are given in Table 1. Deadlines of sensor transactions are firm deadlines. Let $\mathcal{U}$ denote the average processor workload of a set of periodic transactions $\{\tau_i\}_{i=1}^{m}$, i.e., $\mathcal{U} = \sum_{i=1}^{m} \frac{C_i}{P_i}$. The goal of *HH* and *ML* is to determine P_i and D_i $(1 \leq i \leq m)$ such that all the sensor transactions are schedulable and processor workload $\mathcal{U}$ resulting from sensor transactions is minimized. For convenience, we use terms transaction and task interchangeably in this paper.

3.2 Prior Work with Fixed Priority Scheduling Algorithms

In order to guarantee the validity of real-time data in RTDBs, the period and relative deadline of a sensor transaction are each typically set to be one half of the data validity interval in the *Half-Half* approach [15, 11]. The farthest distance (based on the sampling time of a periodic transaction job and the finishing time of its next job) of two consecutive jobs of transaction τ_i is $2P_i$. If $2P_i = \mathcal{V}_i$, then the validity of X_i is guaranteed as long as jobs of τ_i meet their deadlines. Next, we describe the *More-Less* approach that improves the update workload compared to *Half-Half*.

3.2.1 More-Less Using *DM* Algorithm

A set of transactions is *synchronous* if all the first jobs of transactions are initiated at the same time (e.g., time 0). It should be noted that we only discuss *synchronous* transactions in this paper. Given a set of transactions, consider the longest response time for any job of a periodic transaction τ_i. The response time for job $J_{i,j}$ $(j = 1, 2...)$ of transaction τ_i is the difference between the job release time ($r_{i,j}$) and its completion time ($f_{i,j}$).

Lemma 3.1: Given a set of periodic transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ $(D_i \leq P_i)$ that are synchronous, the response time of the first job of τ_i $(1 \leq i \leq m)$ is the longest among all of its jobs in *Deadline Monotonic* (*DM*) scheduling. [14] □

A time instant after which a transaction has the longest response time is called a *critical instant*, e.g., time 0 is a critical instant for all transactions in *DM* if those transactions are *synchronous* [14]. To minimize the update workload, *ML* is used to guarantee temporal consistency with much less processor workload than *HH* [3, 22]. In *ML*, *Deadline Monotonic* (*DM*) is used to schedule periodic sensor transactions. Thus we refer to this scheme as ML_{DM} in the rest of the paper. There are three constraints to respect:

- *Validity constraint*: the sum of the period and relative deadline of transaction τ_i is always less than or equal to $\mathcal{V}_i$, the validity length of the object updated, i.e., $P_i + D_i \leq \mathcal{V}_i$, as shown in Figure 1.
- *Deadline constraint*: the period of a sensor transaction is assigned to be more than half of the validity length of the object to be updated by the transaction, while its corresponding relative deadline is assigned to be less than half of the validity length. 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., $C_i \leq D_i \leq P_i$.
- *Feasibility constraint*: for a given set of sensor transactions, *Deadline Monotonic* scheduling algorithm [14] 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_{DM} assigns priorities to transactions in the *inverse* order of validity length and resolves ties in favor of transactions with larger computation times. It assigns deadlines and periods to τ_i as follows:

$$D_i = f_{i,0} - r_{i,0}, \quad P_i = \mathcal{V}_i - D_i,$$

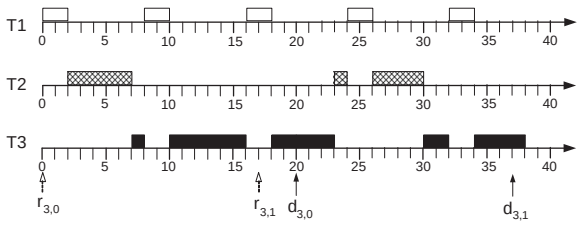

Figure 2. Infeasible schedule of $\mathcal{ML}_{DM}$ ($D_3 > P_3$)

where $f_{i,0}$ and $r_{i,0}$ are finishing and sampling times of $J_{i,0}$, respectively.

3.2.2 ML_{DM} Extension

In $\mathcal{ML}_{DM}$, the first job's response time is the longest response time. This assumption holds for deadline monotonic algorithm when $D_i \leq P_i$. A transaction set is not schedulable if there exists $\tau_i \in \mathcal{T}$, $f_{i,0} - r_{i,0} > \frac{\mathcal{V}_i}{2}$. Next, we discuss whether the restriction of $D_i \leq P_i$ can be relaxed for $\mathcal{ML}_{DM}$, i.e., whether $\mathcal{ML}_{DM}$ can handle arbitrary deadlines.

Example 3.1: Transactions are listed in Table 2 with their transaction number, computation time, and validity interval length. $\mathcal{ML}_{DM}$ is applied to the transaction set, and the resulting deadlines and periods are shown in the Table. Note that the relative deadline of each transaction is the same as its first job's response time in $\mathcal{ML}_{DM}$, and $D_3 > P_3$. Figure 2 depicts the schedule of the transaction set. Job $J_{3,0}$ finishes at its deadline (time 20). Although $J_{3,1}$ is released at time 17, it cannot start until $J_{3,0}$ completes. It starts at time 20, but only completes 8 time units by its deadline (time 37) due to interruption from higher priority transactions. This demonstrates that the first job's response time is not necessarily the longest response time if the deadline is greater than its period. In this case, ensuring that the first jobs' response times are less than their respective deadlines is not sufficient to guarantee the feasibility of $\mathcal{ML}_{DM}$. □

Given a periodic task set with arbitrary deadlines, Lehoczky [12] introduces *level-i busy period* for τ_i, and the longest response time for jobs of τ_i as follows.

Definition 3.2: A level-i busy period is a time interval $[a, b]$ within which jobs of priority i or higher are processed throughout $[a, b]$ but no jobs of priority i or higher are processed in $(a - \epsilon, a)$ or $(b, b + \epsilon)$ for sufficiently small $\epsilon > 0$. □

Lemma 3.2: The longest response time for a job of τ_i occurs during a level-i busy period initiated by a critical instant (e.g., time 0). □

Lemma 3.2 states that the longest response time of τ_i occurs during a level-i busy period, but *this longest response time is not necessarily the response time of the first job of τ_i*. With unknown periods and deadlines, it is not possible to use ML_{DM} to get the longest response time of jobs if a deadline may exceed its period. This is because the first job's response time cannot be used to derive the deadline and period of its corresponding task during a level-i busy period. In [3], a *DM* based approach

is proposed to deal with the case that the first response time is not the longest response time of a task. The idea is to solve a recurrence relation for the response time established during a level-i busy period. Interested readers are referred to [3] for details. In the following, we refer this approach as *Extended ML*$_{DM}$ (i.e., $\mathcal{EML}_{DM}$ for short).

4 Designing *More-Less* Using *EDF*

This section presents a design for temporal consistency maintenance using *EDF* while $D_i \leq P_i$ holds. Section 4.1 formulates an *EDF* optimization problem for $D_i \leq P_i$. Section 4.2 presents feasibility conditions for *EDF* scheduling. Section 4.3 gives the design of *ML* using *EDF* by solving the optimization problem under a sufficient (but not necessary) feasibility condition.

4.1 Restricted Optimization Problem for *ML* Using *EDF*

The optimized solution has to minimize processor workload $\mathcal{U}$ while maintaining the temporal validity of real-time data in RTDBs. This essentially formalizes the following optimization problem that minimizes $\mathcal{U}$ with variables $\vec{P}$ and $\vec{D}$. Note that $\vec{P}$, $\vec{D}$, $\vec{C}$, and $\vec{\mathcal{V}}$ given below are vectors.

Problem 4.1: *Restricted EDF optimization problem:* Given a set of transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ with known $\vec{C}$ and $\vec{\mathcal{V}}$, determine $\vec{P}$ and $\vec{D}$ for synchronous transactions to minimize $\mathcal{U}$, i.e.,

$$\min_{\vec{P},\vec{D}} \mathcal{U}, \text{ where } \mathcal{U} = \sum_{i=1}^{m} \frac{C_i}{P_i} \ (1 \leq i \leq m),$$

subject to:

- *Validity constraint*: $P_i + D_i \leq \mathcal{V}_i$.
- *Deadline constraint*: $C_i \leq D_i \leq P_i$.
- *Feasibility constraint*: $\mathcal{T}$ with derived deadlines and periods is feasible by using *EDF* scheduling. □

As discussed, *deadline constraint* can be further generalized by having $C_i \leq \min(D_i, P_i)$. In this section, *Deadline constraint* in Problem 4.1 is used for our discussion unless specified otherwise. It is generalized in the next section. Next, we consider a sufficient feasibility condition for designing *More-Less* using *EDF*, namely $\mathcal{ML}_{EDF}$. The corresponding feasibility test reduces the complexity of the problem.

4.2 Feasibility Conditions for *EDF*

The periodic task model is a special case of the *sporadic* task model discussed in [1, 2]. A task τ_i in the sporadic task model is characterized by three parameters – an execution time C_i, a deadline D_i, and a minimum separation P_i for the

arrival time of two successive τ_i jobs, with $C_i \leq \min(D_i, P_i)$. If all jobs of a task arrive with the exact minimum separation P_i, this sporadic task becomes a periodic task. Note that the arrival time of a task job is the same as the sampling time of a sensor transaction job.

Given periodic task $\tau_i \in \mathcal{T}(1 \leq i \leq m)$, processor *demand bound function*s, as explained in [1] are defined as follows:

$$\mathcal{H}_i(t) = max(0, (\lfloor \frac{t - D_i}{P_i} \rfloor + 1) \cdot C_i), \quad (1)$$

$$\mathcal{H}_{\mathcal{T}}(t) = \sum_{i=1}^{m} \mathcal{H}_i(t). \quad (2)$$

$\mathcal{H}_i(t)$ represents the processor demand for τ_i on $[0, t)$, i.e., the minimum amount of computation time that must be allocated for τ_i before time t. Similarly, $\mathcal{H}_{\mathcal{T}}(t)$ represents the processor demand for all tasks in $\mathcal{T}$ on $[0, t)$. The following lemma is from Lemma 3 in [1] for a set of periodic transactions $\mathcal{T}$.

Lemma 4.1: $\mathcal{T}$ is feasible iff $\mathcal{H}_{\mathcal{T}}(t) \leq t$ for all $t \geq 0$. □

Lemma 4.1 will be used in Section 5 to derive solutions for *EDF* scheduling. Next, we present a technical result that is a sufficient condition for a set of periodic transactions $\mathcal{T}$ to be feasible using *EDF* scheduler (see [20] for proof).

Lemma 4.2: Given a set of transactions $\mathcal{T}$, if $\sum_{\tau_i \in \mathcal{T}} \frac{C_i}{\min(P_i, D_i)} \leq 1$, then $\mathcal{T}$ is feasible. □

4.3 Designing $\mathcal{ML}_{EDF}$ Using a Sufficient Feasibility Condition

In this section, we investigate the design of $\mathcal{ML}_{EDF}$ using Lemma 4.2 for all τ_i with $D_i \leq P_i$. If Lemma 4.2 is used to derive deadlines and periods for $\mathcal{T}$, the optimization problem for *EDF* scheduling is essentially transformed to the following problem.

Problem 4.2: *EDF optimization problem with sufficient feasibility condition:*

$$\min_{\vec{P}} \mathcal{U}, \text{ where } \mathcal{U} = \sum_{i=1}^{m} \frac{C_i}{P_i} \ (1 \leq i \leq m)$$

subject to:

- *Validity* and *deadline* constraints in Problem 4.1, and
- *Feasibility constraint*: $\sum_{i=1}^{m} \frac{C_i}{D_i} \leq 1$. □

It should be obvious that $\mathcal{U}$ is minimized only if $P_i + D_i = \mathcal{V}_i$. Otherwise, P_i can always be increased and processor utilization can be decreased. Without loss of generality, we assume that

$$D_i = \frac{1}{N_i} \mathcal{V}_i, \quad (3)$$

$$P_i = \frac{N_i - 1}{N_i} \mathcal{V}_i, \quad (4)$$

where $P_i + D_i = \mathcal{V}_i$ and $N_i \geq 2$ in order to satisfy the validity and deadline constraints in Problem 4.2.

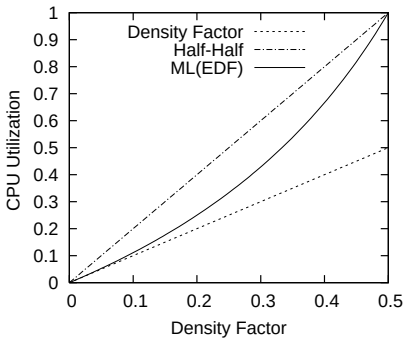

Figure 3. Utilization comparison

Definition 4.1: Given a set of transactions $\mathcal{T}$, the *density factor* of $\mathcal{T}$, denoted as γ, is $\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i}$. □

The following theorem provides a minimal solution for Problem 4.2.

Theorem 4.1: Given the minimization problem in Problem 4.2, there exists a unique minimal solution given by $N_k = \mathcal{N}_{ml}^{opt} = \frac{1}{\gamma}$ $(1 \leq k \leq m \;\&\; 0 < \gamma \leq 0.5)$, which has minimum utilization $\mathcal{U}_{ml}^{opt}(\gamma) = \frac{\gamma}{1-\gamma}$. □

Please refer to Appendix for proofs of all theorems and lemmas. Given a set $\mathcal{T}$ of m sensor transactions, the optimization problem defined in Problem 4.2 can be solved in $O(m)$. By discussion in Section 3, the utilization of *HH* is $\mathcal{U}_{hh}(\gamma) = 2\gamma$.

Theorem 4.2: Given a set $\mathcal{T}$ of sensor transactions ($0 < \gamma \leq 0.5$), $\mathcal{U}_{hh}(\gamma) - \mathcal{U}_{ml}^{opt}(\gamma) \leq 3 - 2\sqrt{2} \approx 0.172$. □

The function curves of γ, $\mathcal{U}_{hh}(\gamma)$ and $\mathcal{U}_{ml}^{opt}(\gamma)$ $(0 < \gamma \leq 0.5)$, are depicted in Figure 3. $\mathcal{ML}_{EDF}$ improves utilization by solving Problem 4.2 in linear time. However, it does not necessarily produce an optimal solution for Problem 4.1 in that the feasibility condition in $\mathcal{ML}_{EDF}$ (i.e., Lemma 4.2) is sufficient but not necessary. There exist *EDF* feasibility conditions that are both sufficient and necessary [1, 2, 16]. For example, [16] presents a pseudo-polynomial algorithm only for $C_i \leq D_i \leq P_i$, whereas Lemma 4.1 [1] can be applied to any arbitrary deadlines. Next, we present our integer programming based search algorithms, for assigning periods and deadlines using *EDF* scheduling with relaxed *deadline constraint*, i.e., $C_i \leq \min(D_i, P_i)$.

5 Designing *Search* Algorithms Using *EDF*

This section presents algorithms – using the sufficient and necessary feasibility condition for *EDF* in Lemma 4.1 – to *search* for optimal periods and deadlines of sensor transactions when *arbitrary* deadlines are allowed. Section 5.1 defines a general *EDF* optimization problem with relaxed *deadline constraint*, i.e., $C_i \leq \min(D_i, P_i)$. Section 5.2 shows the optimality of *EDF* solutions compared to solutions produced by other periodic schedulers. Sections 5.3 and 5.4 present *branch and bound* based optimal and heuristic search algorithms, respectively, and show that the general problem can be solved efficiently without reducing schedulability.

5.1 Formalizing the General *EDF* Optimization Problem

Following Lemma 4.1, Problem 4.1 can be generalized to the following problem.

Problem 5.1: *General EDF optimization problem:*

$$\mathcal{U}_{EDF}^{opt} = \min_{\vec{P},\vec{D}} \mathcal{U}, \text{ where } \mathcal{U} = \sum_{i=1}^{m} \frac{C_i}{P_i}$$

subject to:

- *Validity constraint*: $P_i + D_i \leq \mathcal{V}_i$.
- *Deadline constraint*: $C_i \leq \min(D_i, P_i)$.
- *Feasibility constraint*: $\forall t, \mathcal{H}_{\mathcal{T}}(t) \leq t$. □

The problem of deciding whether sporadic task set $\mathcal{T}$ with arbitrary deadlines (i.e., $C_i \leq \min(D_i, P_i)$) is feasible with given deadlines and periods is known to be in *co-NP* [1]. The feasibility test for $\mathcal{T}$ with given deadlines and periods is a sub-problem of the optimization problem in Problem 5.1, i.e., Problem 5.1 contains a known *co-NP* problem.

By definition of Eq. 1 and 2,

$$\mathcal{H}_{\mathcal{T}}(t) = \sum_{i=1}^{m} max(0, (\lfloor \frac{t - D_i}{P_i} \rfloor + 1) \cdot C_i). \tag{5}$$

Note that the optimal solution can only be achieved when $P_i + D_i = \mathcal{V}_i$. Thus, $D_i = \mathcal{V}_i - P_i$, and considering the *deadline constraint* in Problem 5.1, we derive the *period constraint* as follows by replacing D_i with $\mathcal{V}_i - P_i$ in the *deadline constraint*:

$$C_i \leq P_i \leq \mathcal{V}_i - C_i. \tag{6}$$

Replacing D_i with $\mathcal{V}_i - P_i$ in Eq. 5, the *feasibility constraint* in Problem 5.1 is:

$$\mathcal{H}_{\mathcal{T}}(t) = \sum_{i=1}^{m} max(0, (\lfloor \frac{(t - \mathcal{V}_i)}{P_i} \rfloor + 2) \cdot C_i) \leq t. \tag{7}$$

Eq. 7 is also referred as *time constraint*. Thus, Problem 5.1 can be transformed into minimizing $\mathcal{U}$ subject to the *period constraint* (Eq. 6) and *time constraint* (Eq. 7). There are no existing solutions for this problem, which is difficult since it has an unbounded variable t. Next, we present a lemma that gives a bound on the "time length" necessary for determining the feasibility.

Lemma 5.1: Given a set of transactions $\mathcal{T}$ with $\mathcal{U} < 1$, let

$$t_{\mathcal{B}}(\vec{P}) = \max(\max_i(\mathcal{V}_i - 2C_i), \frac{\sum_{i=1}^{m}(2 - \frac{\mathcal{V}_i}{P_i})C_i}{1 - \mathcal{U}}).$$

$\mathcal{T}$ is feasible iff $\forall t < t_{\mathcal{B}}(\vec{P}), \mathcal{H}_{\mathcal{T}}(t) \leq t$. □

Lemma 5.1 indicates that a feasibility testing algorithm does not have to check $\forall t \leq P, \mathcal{H}_{\mathcal{T}}(t) \leq t$, where P is the least common multiple of all periods. Thus a feasibility testing algorithm based on Lemma 5.1 can run in *pseudo-polynomial* time for a large percentage of sensor transaction sets, although it is exponential in the worst-case. Its complexity is similar to the feasibility testing algorithms in [1, 16]. Note that Lemma 5.1 can only be applied when $\vec{P}$ is known. It cannot be applied to Problem 5.1 for feasibility test unless $\vec{P}$ has been determined.

5.2 Optimality of *EDF* Solutions

EDF is not the only scheduler that can be used to schedule periodic tasks. One open question is whether an optimal solution for Problem 5.1 minimizes utilization for all schedulers that can be used to derive periods and deadlines of $\mathcal{T}$. Problem 5.1 can be further generalized as follows.

Problem 5.2: *General scheduler optimization problem:*

$$\min_{\vec{P},\vec{D}} \mathcal{U}, \text{ where } \mathcal{U} = \sum_{i=1}^{m} \frac{C_i}{P_i}$$

subject to:

- *Validity* and *deadline* constraints in Problem 5.1.
- *Feasibility constraint*: a periodic schedule from any scheduler that is feasible for periodic transaction set $\mathcal{T}$. □

The following theorem shows optimality of *EDF* solutions.

Theorem 5.1: An optimized utilization of *EDF* solutions, $\mathcal{U}_{EDF}^{opt}$, of Problem 5.1 is also optimal for Problem 5.2 in that if there exists a periodic schedule that is feasible for Problem 5.2 with utilization $\mathcal{U}$, then $\mathcal{U}_{EDF}^{opt} \leq \mathcal{U}$. □

Next, we present a search algorithm, which uses the *branch and bound* method in integer programming, to find an optimal solution for *EDF* scheduling.

5.3 $\mathcal{OS}_{EDF}$: Optimal Search Using *EDF*

This subsection presents our *optimal search* algorithm using *EDF* scheduling, namely $\mathcal{OS}_{EDF}$. It finds optimal $\vec{P}$ that minimizes processor utilization in Problem 5.1. We first give the high-level algorithm, then define its constituents.

$\mathcal{OS}_{EDF}$ is depicted in Algorithm 5.1. It first relaxes Problem 5.1 by defining a *proxy* problem that initially has no time constraint. It then solves the proxy problem and obtains a trial solution $\vec{P}^K$ $(K = 0, 1, 2, ..)$. A *testing* problem is defined to test if $\vec{P}^K$ satisfies all time constraints. If not, a new time constraint is added to tighten the proxy problem. Then the algorithm iterates to solve the proxy problem again. This iteration continues until the trial solution $\vec{P}^K$ satisfies all time constraints, or the proxy problem becomes unsolvable.

Algorithm 5.1 $\mathcal{OS}_{EDF}$**:**

1. *Define a relaxed version of Problem 5.1, namely the* proxy *problem (see Problem 5.3). Problem 5.3 has no time constraint in the first iteration. A new time constraint is added at each iteration. Therefore, Problem 5.3 has only a limited number of time constraints in each iteration.*

2. *In the* $K^{th}(K = 0, 1, 2, ...)$ *iteration, solve Problem 5.3 to obtain a trial solution* $\vec{P}^K$*. Since Problem 5.3 does not have all time constraints (in Eq. 7), the utilization of its solution,* $\mathcal{U}^K$*, is no greater than that of the optimal solution to Problem 5.1,* $\mathcal{U}^{opt}_{EDF}$*.*

3. *Given trial solution* $\vec{P}^K$*, solve the* testing *problem (see Problem 5.4). This determines whether any time constraint (in Eq. 7) is violated. If so, the solution to Problem 5.4 identifies a time point,* t_K*, at which the time constraint is the tightest for the given* $\vec{P}^K$ *(i.e.,* $\vec{P}^K$ *exceeds the constraint to the largest extent at* t_K*). Then Problem 5.3 is updated by adding a new time constraint at* t_K*. The trial solution to the updated Problem 5.3 in the next iteration is feasible for more time constraints (in Eq. 7), or the problem becomes unsolvable.*

4. *Continue for Steps 2 and 3 until Problem 5.3 produces a solution that violates no time constraint in Problem 5.1, as determined by Problem 5.4.*

The *proxy* problem in Algorithm 5.1 is defined as follows:

Problem 5.3: *Proxy problem*:

$$\mathcal{U}^K = \min_{\vec{z}} \sum_{i=1}^{m} C_i \sum_{j=C_i}^{\mathcal{V}_i - C_i} \frac{z_{i,j}}{j} \tag{8}$$

subject to:

$$z_{i,j} \in \{0, 1\} \quad \text{and} \quad \sum_{j=C_i}^{\mathcal{V}_i - C_i} z_{i,j} = 1, \tag{9}$$

$$\sum_{i=1}^{m} C_i \sum_{j=C_i}^{\mathcal{V}_i - C_i} z_{i,j} \cdot \max\{0, \lfloor \frac{t_k - \mathcal{V}_i}{j} \rfloor + 2\} \leq t_k, \tag{10}$$

where i, j, and k are integers, and $i = 1, .., m,\ j = C_i, .., \mathcal{V}_i - C_i,\ k = 1, .., K$. □

$\mathcal{U}^K$ is the minimized processor utilization at the K^{th} iteration. Binary variable $z_{i,j}$ determines the value of P_i between C_i and $\mathcal{V}_i - C_i (1 \leq i \leq m)$. Constraint 9 indicates that one and only one $z_{i,j}$ can be set to 1 for each τ_i. This implies that $P_i = j$ for $z_{i,j} = 1$ $(C_i \leq j \leq \mathcal{V}_i - C_i)$. Note that

$$\sum_{j=C_i}^{\mathcal{V}_i - C_i} j z_{i,j} = P_i, \tag{11}$$

$$\sum_{j=C_i}^{\mathcal{V}_i - C_i} \frac{z_{i,j}}{j} = \frac{1}{P_i}. \tag{12}$$

Constraint 10 satisfies time constraints (Eq. 7) at selected time points $t = t_1, ..., t_K$. In the first iteration (i.e., $K = 0$), no time constraint is present. K is incremented in each iteration afterwards. Hence, Problem 5.3 only has K time constraints as opposed to an infinite number of time constraints in Problem 5.1.

Lemma 5.2 demonstrates a relationship between Problems 5.1 and 5.3. It shows that the optimal utilization of Problem 5.3 for each iteration $(K = 1, 2, ...,)$ constitutes a monotonically increasing sequence that is bounded by the optimal utilization of Problem 5.1.

Lemma 5.2: Let $\vec{z}^K$ be an optimal solution to Problem 5.3 in the K^{th} iteration. If $\mathcal{U}^K$ is the minimized utilization in Eq. 8 where $z_{i,j} = z_{i,j}^K$ $(1 \leq i \leq m,\ C_i \leq j \leq \mathcal{V}_i - C_i)$, then

$$\mathcal{U}^0 \leq \mathcal{U}^1 ... \leq \mathcal{U}^K \leq \mathcal{U}_{EDF}^{opt},$$

where $\mathcal{U}_{EDF}^{opt}$ is the optimized utilization to Problem 5.1. □

Lemma 5.2 implies that if $\vec{z}^K$ satisfies the time constraint at any $t \geq 0$, then the corresponding $\vec{P}^K$ must be an optimal solution to Problem 5.1. Hence the solution obtained with Alg. 5.1 has optimal utilization.

Considering Eq. 7, we define

$$F(t) = t - \sum_{i=1}^{m} \max(0, (\lfloor \frac{t - \mathcal{V}_i}{P_i} \rfloor + 2)C_i).$$

To determine whether $\vec{P}^K$ violates any time constraint, we examine the minimum $F(t)$ in the K^{th} iteration, namely F^K. The time constraint is violated if $F^K < 0$. This is formulated as an integer programming problem, namely the *testing problem*.

Problem 5.4: *Testing problem*:

$$F^K = \min_{t, \vec{y} \geq 0} [t - \sum_{i=1}^{m} \max(0, (y_i + 2)C_i)] \tag{13}$$

subject to:

$$\frac{(t - \mathcal{V}_i)}{P_i^K} - 1 \leq y_i \leq \frac{(t - \mathcal{V}_i)}{P_i^K} \ \ (1 \leq i \leq m), \tag{14}$$

where t and $\vec{y}$ are integer variables, and $\vec{P}^K$, the trial solution from Problem 5.3 in the same iteration, is an input parameter. Variable $\vec{y}$ is introduced to remove the floor function in the definition of $F(t)$. □

Lemma 5.3 gives the rationale for formulating and solving Problem 5.4.

Lemma 5.3: Let $\vec{z}^K$ optimize Problem 5.3 in the K^{th} iteration, and

$$P_i^K = \sum_{j=C_i}^{\mathcal{V}_i - C_i} j z_{i,j}, \quad i = 1, .., m.$$

Then $\vec{P}^K$ is the optimal solution to Problem 5.1 if and only if $F^K \geq 0$. □

i	C_i	$\mathcal{V}_i$	$\mathcal{OS}_{EDF}$	
			P_i	D_i
1	1	5	4	1
2	3	15	11	4
3	6	30	14	16

Table 3. Example Parameters and Derived P_i and D_i for Illustration ($D_3 > P_3$)

	Problem 5.3		Problem 5.4	
K	$\mathcal{U}^K$	$\vec{P}^K$	t_K	F^K
0	0.751	$(4, 12, 24)$	6	-5
1	0.760	$(4, 12, 23)$	7	-4
2	0.773	$(4, 12, 22)$	8	-3
.....	.....	....	...	....
8	0.951	$(4, 11, 14)$	16	0

Table 4. $\mathcal{OS}_{EDF}$ Iterative Process

It is noteworthy that Lemma 5.3 defines a stopping criterion, i.e., $F^K \geq 0$, for terminating the computation when an optimal solution is found. Further, if $F^K < 0$, then the trial solution is infeasible for Problem 5.1. By definition of Eq. 13, the time at which F^K reaches the minimum, denoted by t_K, is the tightest time constraint where $\vec{P}^K$ exceeds the constraint to the largest extent. We add the time constraint at t_K to Problem 5.3 in the next iteration.

Since there are only limited choices for $z_{i,j}$ (and thus P_i) in Problem 5.3, the iteration in Algorithm 5.1 ends up in one of two following situations: (1) F^K in Problem 5.4 becomes non-negative, at which point an optimal solution is found (by Lemma 5.3); (2) as more time constraints are added to Eq. 10, Problem 5.3 eventually becomes infeasible. This implies that there exists a subset of time constraints in Eq. 10 (thus, in Eq. 7) that cannot be satisfied simultaneously. Note that $\mathcal{OS}_{EDF}$ transforms Problem 5.1 to Problems 5.3 and 5.4, which are linear integer programming models that can be solved by the *branch and bound* method [21] provided by commercial optimization software, e.g., *CPLEX* [1].

Illustration: We demonstrate how Alg. 5.1 works with the example in Table 3. Table 4 depicts the iterative process for finding an optimal solution for the transaction set in Table 3. Starting from $K = 0$ (i.e., the case of no time constraint), the solution to Problem 5.3 is $\vec{P}^0 = (4, 12, 24)$. Solving the testing problem (Problem 5.4), $F^0 = -5$. By Lemma 5.3, the solution violates the time constraint. The tightest time constraint occurs at $t = 6$, at which $F^0 = -5$. Therefore, the following time constraint at $t_0 = 6$ is added to Problem 5.3:

$$\sum_{i=1}^{m} C_i \sum_{j=C_i}^{\mathcal{V}_i - C_i} z_{i,j} \max\{0, \lfloor \frac{6 - \mathcal{V}_i}{j} \rfloor + 2\} \leq 6.$$

Solving Problem 5.3 with updated constraints results in another solution $\vec{P}^1 = (4, 12, 23)$. The corresponding solution of the testing problem is $F^1 = -4$. The new trial solution $\vec{P}^1$ again violates time constraint (Eq. 7). The violation leads to another addition of time constraint at $t_1 = 7$ to Problem 5.3. The algorithm continues for 7 more iterations until it obtains a solution $\vec{P}^8 = (4, 11, 14)$, which is the optimal solution to Problem 5.1 because $F^8 = 0$. Note that the example transactions in Table 3 are feasible with neither $\mathcal{ML}_{EDF}$ nor $\mathcal{ML}_{DM}$.

[1] http://www.ilog.com/products/cplex

5.4 $\mathcal{HS}_{EDF}$: Heuristic Search Using *EDF*

While $\mathcal{OS}_{EDF}$ guarantees optimality, its applicability is limited by the problem size. For each task, there are $\mathcal{V}_i - 2C_i + 1$ binary variables $z_{i,j}$ in Problem 5.3 as index j is from C_i to $\mathcal{V}_i - C_i$. If the number of tasks and values of $\mathcal{V}_i - 2C_i$ are large, then the model can involve a large number of binary variables, and the problem becomes difficult to solve. This is verified in our experiments.

This section presents our heuristic algorithm, namely $\mathcal{HS}_{EDF}$, which is more efficient for solving Problem 5.1. $\mathcal{HS}_{EDF}$ is based on the following rationale.

1. The objective function $\mathcal{U}$ (in Problem 5.1) is a strictly decreasing function of P_i. Moreover, for $t \geq \mathcal{V}_i$,

$$\mathcal{H}_i(t) = \max\{0, (\lfloor (t - \mathcal{V}_i)/P_i \rfloor + 2)C_i\} \tag{15}$$

   decreases as P_i increases. The larger the value of P_i $(1 \leq i \leq m)$, the easier to satisfy Eq. 7. Therefore, at the beginning of $\mathcal{HS}_{EDF}$, P_i is set to be its largest possible value, $\mathcal{V}_i - C_i$ $(i = 1, .., m)$.

2. Under the period constraint (Eq. 6), the initial solution drives $\mathcal{U}$ to the minimum, which thus is the optimal solution if it also satisfies the time constraint (Eq. 7) at all t. On the other hand, for $t \geq \mathcal{V}_i (i = 1, .., m)$, it is impossible to reduce $\mathcal{H}_i(t)$, since doing so requires increasing P_i above its upper bound $\mathcal{V}_i - C_i$. Therefore, if the initial solution violates the time constraint (Eq. 7) at t for $t \geq \mathcal{V}_{max}$, where

$$\mathcal{V}_{max} = \max_i(\mathcal{V}_i), \tag{16}$$

   then the problem is infeasible.

3. Lemma 5.1 is used to test if a given $\vec{P}$ violates any time constraint. Specifically, given $\vec{P}$, we calculate $t_{\mathcal{B}}(\vec{P})$ defined in the lemma. For each $t \leq t_{\mathcal{B}}(\vec{P})$, we evaluate $\mathcal{H}_i(t)$ according to Eq. 15. The summation of $\mathcal{H}_i(t)$ gives $\mathcal{H}_{\mathcal{T}}(t)$, which should not exceed t. Otherwise the constraint Eq. 7 is infeasible under the current value of $\vec{P}$ and an adjustment is required. Note that once $\vec{P}$ is changed, the value of $t_{\mathcal{B}}(\vec{P})$ is changed correspondingly.

4. Suppose the test by Lemma 5.1 shows that the initial solution violates the time constraint (Eq. 7) at t', i.e., $\mathcal{H}_{\mathcal{T}}(t') > t'$. In this case, given set $\{\tau_i : C_i \leq t' < \mathcal{V}_i - C_i\}$ and $\mathcal{H}_i(t')$ definition (Eq. 15),

$$\mathcal{H}_i(t') = \begin{cases} 0, & C_i \leq P_i < \mathcal{V}_i - t', \\ C_i, & \mathcal{V}_i - t' \leq P_i \leq \mathcal{V}_i - C_i \end{cases} \tag{17}$$

   For transaction τ_i that satisfies $\mathcal{H}_i(t') = C_i$, $\mathcal{H}_i(t')$ can be reduced to 0 by setting $P_i = \mathcal{V}_i - t' - 1$ if $\mathcal{V}_i - t' - 1 \geq C_i$.

Otherwise it violates the period constraint (Eq. 6) at the lower end. Note that in $\mathcal{H}_i(t')$,

$$\lfloor \frac{t' - \mathcal{V}_i}{P_i} \rfloor \begin{cases} \geq 0 & t' \geq \mathcal{V}_i, \\ = -1 & P_i \geq \mathcal{V}_i - t' \ \& \ t' < \mathcal{V}_i, \\ \leq -2 & P_i \leq \mathcal{V}_i - t' \ \& \ t' < \mathcal{V}_i. \end{cases} \tag{18}$$

These three cases of $\lfloor \frac{t' - \mathcal{V}_i}{P_i} \rfloor$ are discussed below.

(a) $\lfloor \frac{t' - \mathcal{V}_i}{P_i} \rfloor \geq 0$ implies that $\mathcal{H}_i(t') \geq 2C_i$. Reducing P_i increases $\mathcal{H}_i(t')$. This does not help reduce $\mathcal{H}_\mathcal{T}(t')$. Thus, P_i should not be changed.

(b) $\lfloor \frac{t' - \mathcal{V}_i}{P_i} \rfloor = -1$ implies that $\mathcal{H}_i(t') = C_i$. If P_i is reduced to $\mathcal{V}_i - t' - 1 \geq C_i$, then $\lfloor \frac{t' - \mathcal{V}_i}{P_i} \rfloor = -2$. In this case, $\mathcal{H}_i(t')$ is reduced to 0 from C_i. Thus, $\mathcal{H}_\mathcal{T}(t')$ is reduced.

(c) $\lfloor \frac{t' - \mathcal{V}_i}{P_i} \rfloor \leq -2$ implies that $\mathcal{H}_i(t') = 0$. If P_i is reduced, $\mathcal{H}_i(t')$ stays 0 but $\mathcal{U}$ is increased. So P_i should not be changed.

Thus, P_i can only be changed in case (b). Let

$$\mathcal{R}(t') = \{\tau_i : \lfloor \frac{t' - \mathcal{V}_i}{P_i} \rfloor = -1 \ \& \ \mathcal{V}_i - t' - 1 \geq C_i\} \tag{19}$$

be the set of transactions whose periods can be reduced to $\mathcal{V}_i - t' - 1$. Note that $\mathcal{V}_i - t' - 1 \geq C_i$ implies that

$$t' \leq \mathcal{V}_i - C_i - 1 \leq \mathcal{V}_i.$$

If $\mathcal{R}(t')$ has a sufficient number of transactions, then reducing periods of some transactions may satisfy the time constraint (Eq. 7) at t'.

5. Nevertheless, reducing P_i always increases $\mathcal{U}$. Specifically, if P_i ($\tau_i \in \mathcal{R}(t')$) is reduced to $\mathcal{V}_i - t' - 1$, then $\mathcal{U}$ is increased by the following positive amount:

$$\delta_i \equiv \frac{C_i}{\mathcal{V}_i - t' - 1} - \frac{C_i}{P_i}.$$

Reducing P_i not only affects the objective function $\mathcal{U}$ negatively, but also increases $\mathcal{H}_i(t)$ for time $t > \mathcal{V}_i$, which tightens the time constraint (Eq. 7) at those points. Therefore, care needs to be taken to decide which transactions' periods in $\mathcal{R}(t')$ should be reduced so that $\mathcal{H}_\tau(t)$ is not overly increased for $t > \mathcal{V}_i$, thereby maintaining $\mathcal{H}_\tau(t) \leq t$. In $\mathcal{HS}_{EDF}$, transactions are selected from $\mathcal{R}(t')$ for reducing their periods by solving the following *selection problem*.

Problem 5.5: *Selection Problem:*

$$\min_{w_i:\ \tau_i \in \mathcal{R}(t')} \sum_{i=1}^{m} w_i \delta_i \tag{20}$$

subject to:

$$w_i \in \{0,1\} \quad \& \sum_{i:\tau_i \in \mathcal{R}(t')} C_i w_i \geq \mathcal{H}_{\mathcal{T}}(t') - t', \tag{21}$$

where $w_i : \tau_i \in \mathcal{R}(t')$ are binary variables. □

By solving Problem 5.5, P_i $(\tau_i \in \mathcal{R}(t'))$ is reduced to $\mathcal{V}_i - t' - 1$ if $w_i = 1$, or unchanged if $w_i = 0$. Thus periods are reduced in a manner that the increment of $\mathcal{U}$ is minimized. Eq. 21 eliminates the deficit of the time constraint, $\mathcal{H}_{\mathcal{T}}(t') - t'$, at t', which makes Eq. 7 feasible at t', i.e., $\mathcal{H}_{\mathcal{T}}(t') \leq t'$. We consider Problem 5.1 infeasible if Problem 5.5 is unsolvable. Each instance of Problem 5.5 is a *Knapsack* model that can be solved by *CPLEX*.

Algorithm 5.2 $\mathcal{HS}_{EDF}$**:**

1. *Initialization:* $P_i = \mathcal{V}_i - C_i,\ i = 1, .., m$. *With the initial solution, the time constraint is always satisfied at* $t = 0$ *as* $\mathcal{H}_i(0) = 0$. *Set the starting time* $t = 1$.

2. *Calculate utilization* $\mathcal{U}(\vec{P})$ *(Problem 5.1). Stop if* $\mathcal{U}(\vec{P}) > 1$, *the problem is infeasible.*

3. *Calculate* $t_{\mathcal{B}}(\vec{P})$ *as given by Lemma 5.1.*

   If $t = t_{\mathcal{B}}(\vec{P})$, *stop and return* $\vec{P^*} = \vec{P}$ *as the final solution.*

4. *If* $\mathcal{H}_{\mathcal{T}}(t) \leq t$, *which means that* $\vec{P}$ *does not violate Eq. 7 at* t, *set* $t = t + 1$ *and go to Step 3. Otherwise, solve Problem 5.5, reset* $P_i = \mathcal{V}_i - t - 1$ *for* $\tau_i \in \mathcal{R}(t)$ *with* $w_i = 1$, *and go to Step 2. If Problem 5.5 is unsolvable, then stop – Problem 5.1 is infeasible.*

One salient feature of $\mathcal{HS}_{EDF}$ is that it only checks each time point once: if the algorithm reaches time t', it is not necessary to check feasibility for $t < t'$ even if $\vec{P}$ has been changed at t'. This property is supported by Theorem 5.2.

Theorem 5.2: Let $\mathcal{T}(t')$ be the transaction set with solution $\vec{P}(t')$ obtained after Step 4 in Alg. 5.2 at time $t' \geq 0$. Then $\mathcal{T}(t')$ with $\vec{P}(t')$ satisfies the time constraint at all $t \leq t'$. That is, let

$$\mathcal{H}_{\mathcal{T}(t')}(t) = \sum_{i=1}^{m} max(0, (\lfloor \frac{(t - \mathcal{V}_i)}{P_i(t')} \rfloor + 2) \cdot C_i),$$

then $\mathcal{H}_{\mathcal{T}(t')}(t) \leq t$ for $t \leq t'$. □

Theorem 5.2 indicates only one pass is needed for each time point t in Alg. 5.2. The selection problem is solved when the current solution violates the time constraint at t and $\mathcal{R}(t)$ is not empty, which requires $t \leq \mathcal{V}_{max}$ $(Eq.16)$. Therefore, Alg. 5.2 at most solves the selection problem $\mathcal{V}_{max}$ times, although it may iterate $t_{\mathcal{B}}(\vec{P^*})$ time points where $\vec{P^*}$ is the final solution of Alg. 5.2. At each iteration, Alg. 5.2 tests feasibility condition $\mathcal{H}_{\mathcal{T}}(t) \leq t$.

Illustration: To illustrate $\mathcal{HS}_{EDF}$, the algorithm is applied to the example in Table 3. Table 5 shows the iteration process. At the starting point, $\vec{P} = \vec{\mathcal{V}} - \vec{C} = (4, 12, 24)$. As mentioned above, the solution always satisfies Eq. 7 at $t = 0$, so the

algorithm starts at $t = 1$. The time constraint is first violated at $t = 3$, $\mathcal{H}_{\mathcal{T}}(t)$ exceeds t by 1. Applying $\vec{P}$ and $t' = 3$ to Eq. 19, $\mathcal{R}(t') = \{\tau_1, \tau_2\}$. At Step 4 of the algorithm,

$$P_1' = \mathcal{V}_1 - t - 1 = 1, \;\; P_2' = \mathcal{V}_2 - t - 1 = 11,$$

$$\delta_1 = C_1(1/P_1' - 1/P_1) = 0.75, \;\; \delta_2 = C_2(1/P_2' - 1/P_2) = 0.022.$$

This produces the following instance of Problem 5.5:

$$\min_{w_1, w_2} \{0.75w_1 + 0.022w_2 \; : \; w_1 + 3w_2 \geq 1\}, \tag{22}$$

The solution is $w_1 = 0, w_2 = 1$. P_2 is changed to 11 and P_1 remains the same. It then proceeds to $t = 4$. Once $\vec{P}$ is changed, $t_{\mathcal{B}}(\vec{P})$ also needs to be updated.

As t increments and the iteration continues, either Eq. 7 ($\mathcal{H}_{\mathcal{T}}(t) \leq t$) is verified to hold at t, in which case $\vec{P}$ is unchanged, or the time constraint is violated and $\vec{P}$ is updated. In Table 5, an asterisk is added as superscript in the first column for the latter case. In these cases, $\mathcal{H}_{\mathcal{T}}(t) - t$ in the fourth column is calculated before the change of the solution while $\vec{P}$ in the second column is the new solution. Overall, $\vec{P}$ has been changed eight times before $t = 15$ at which point $\vec{P} = (4, 11, 14)$. From time 15 to $t_{\mathcal{B}}(\vec{P}) = 38$, Eq. 7 is satisfied and $\vec{P}$ remains unchanged. Following Lemma 5.1, no time constraint can be violated if $t > t_{\mathcal{B}}(\vec{P})$ because $\sum_{i=1}^{m} C_i/P_i < 1$. Therefore, the algorithm stops at $t = 38$. Solution $(4, 11, 14)$ is the same as that obtained with $\mathcal{OS}_{EDF}$. This demonstrates that $\mathcal{HS}_{EDF}$ and $\mathcal{OS}_{EDF}$ can schedule a larger set of sensor transactions than existing approaches.

$\mathcal{OS}_{EDF}$ **vs.** $\mathcal{HS}_{EDF}$: Table 6 compares numbers of variables (Var#) and constraints (Cons#) in Problems 5.3, 5.4, and 5.5. For $\mathcal{OS}_{EDF}$, the left number in a parenthesis represents Problem 5.3, while the right one represents Problem 5.4. The table also compares numbers of iterations (Iter#) in Alg. 5.1 and 5.2 [2] and along with optimization problem instances solved by those algorithms (Inst#). $\mathcal{OS}_{EDF}$ has significantly larger numbers of variables and constraints than $\mathcal{HS}_{EDF}$. This is why $\mathcal{HS}_{EDF}$ is more efficient than $\mathcal{OS}_{EDF}$, and $\mathcal{OS}_{EDF}$ does not scale well with the problem size. The comparisons of numbers of iterations and solved optimization problem instances are problem dependent. Note that $\mathcal{HS}_{EDF}$ still involves solving the Knapsack problem with the *branch and bound* method. Nevertheless, the number of variables is linear to the number of transactions.

Optimization for τ_i with $D_i > P_i$: Both Alg. 5.1 and 5.2 enable *EDF* scheduling for transactions with deadline *larger* than period. Given transaction τ_i $(1 \leq i \leq m)$ with D_i and P_i assigned by Alg. 5.2 (or Alg. 5.1), if $D_i > P_i$, it is possible to skip certain transaction jobs given the following lemma:

Lemma 5.4: (**Job Skipping**) Given transaction τ_i $(1 \leq i \leq m)$ with $D_i > P_i$ derived from Alg. 5.2, and jobs $J_{i,j}$ and $J_{i,j+1}$ $(j \geq 0)$, if $J_{i,j}$ cannot be started before $r_{i,j+1}$, then $J_{i,j}$ can be skipped and $J_{i,j+1}$ can be executed in $[r_{i,j+1}, d_{i,j}]$ while the validity constraint is guaranteed. □

[2] Note that $\vec{P}^*$ is the final solution in Alg. 5.2.

t	$\vec{P}$	$\mathcal{U}$	$\mathcal{H}_{\mathcal{T}}(t) - t$	$t_{\mathcal{B}}(\vec{P})$
1	$(4, 12, 24)$	0.750	0	31
2	$(4, 12, 24)$	0.750	-1	31
3^*	$(4, 11, 24)$	0.773	1	32
4	$(4, 11, 24)$	0.773	0	32
.....	.....	....	...	....
15^*	$(4, 11, 14)$	0.951	1	38
.....	.....	....	...	....
38	$(4, 11, 14)$	0.951	-4	38

Table 5. $\mathcal{HS}_{EDF}$ **Iterative Process**

Alg.	$\mathcal{OS}_{EDF}$	$\mathcal{HS}_{EDF}$
Variable#	$(\sum_{i=1}^{m} (\mathcal{V}_i - 2C_i + 1), m)$	m
Constraint#	$(K + m, m)$	1
Iteration#	K	$t_{\mathcal{B}}(\vec{P^*})$
Instance#	K	$\mathcal{V}_{max}$

Table 6. $\mathcal{OS}_{EDF}$ **and** $\mathcal{HS}_{EDF}$ **Comparison**

6 Performance Evaluation

This section presents important results from our experimental studies of the proposed $\mathcal{HS}_{EDF}$ and $\mathcal{ML}_{EDF}$ algorithms.

6.1 Simulation Model and Parameters

We have conducted experiments to compare the performance of $\mathcal{HS}_{EDF}$ and $\mathcal{ML}_{EDF}$. In our experiments, we compare the update transaction workloads produced by $\mathcal{HS}_{EDF}$ and $\mathcal{ML}_{EDF}$. It is demonstrated that $\mathcal{HS}_{EDF}$ produces lower CPU workload than $\mathcal{ML}_{EDF}$.

A summary of the parameters and default settings used in experiments are presented in Table 7. The baseline values for the parameters follow those used in [22], which are originally from air traffic control applications. For system configurations, we only consider a single CPU, main memory based RTDBS. The number of real-time data objects is uniformly varied from 50 to 300 and it is assumed that the validity interval length of each real-time data object is uniformly varied from 4000 to 8000 ms. For update transactions, it is assumed that each transaction updates one real-time data object, and the CPU time for each transaction is uniformly varied from 5 to 15 ms.

Parameter Meaning	Value
No. of CPU	1
No. of real-time data objects	[50, 300]
Validity interval of data objects (ms)	[4000, 8000]
CPU time per data access (ms)	[5, 15]
Update transaction length	1

Table 7. Experimental Parameters and Settings

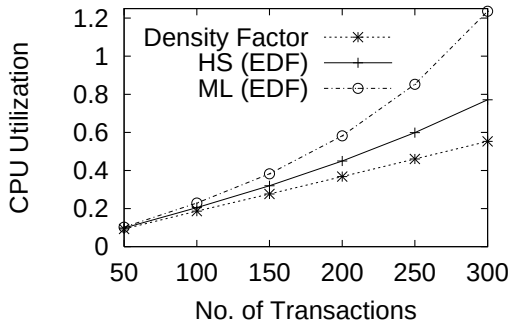

Figure 4. $\mathcal{ML}_{EDF}$ **vs.** $\mathcal{HS}_{EDF}$

6.2 Experimental Results

In our experiments, Alg. 5.2 is investigated with sensor update transaction sets, and its results are depicted in Figure 4, in which the x-axis is the number of update transactions and the y-axis is the average CPU workload. Results of $\mathcal{HS}_{EDF}$ are compared with those of $\mathcal{ML}_{EDF}$. The density factor, which provides a lower bound of the utilization, is also plotted.

For tested data sets, $\mathcal{HS}_{EDF}$ consistently outperforms $\mathcal{ML}_{EDF}$. Its advantage becomes increasingly obvious when the number of transactions gets larger. In particular, the set of 300 transactions cannot be scheduled by $\mathcal{ML}_{EDF}$, but it is schedulable under $\mathcal{HS}_{EDF}$. We have found out, through our experiments, the major reason why $\mathcal{HS}_{EDF}$ outperforms $\mathcal{ML}_{EDF}$ in terms of CPU utilization is due to the more accurate schedulability condition in Lemma 4.1. We have also done experiments with parameter settings different from Table 7. The results are similar to what is depicted in Figure 4.

Other Experiments: We conducted experiments for the comparison of $\mathcal{ML}_{DM}$, $\mathcal{EML}_{DM}$ and $\mathcal{HS}_{EDF}$. We found out that these algorithms produce about the same CPU workload if the set of sensor transactions are schedulable by all the algorithms, but $\mathcal{HS}_{EDF}$ and $\mathcal{EML}_{DM}$ can schedule a slightly larger set of sensor transactions due to the fact that $\mathcal{HS}_{EDF}$ allows arbitrary deadlines. However, it is difficult to quantify how much they can improve the feasibility of $\mathcal{ML}_{DM}$ as the workload generation plays an important role in such a comparison. This issue needs further investigation, which is left as

future work.

We also conducted another set of experiments by replacing the deadline constraint in Problem 5.1 with $C_i \leq D_i \leq P_i$, i.e., deadlines are not greater than their corresponding periods. Then the $\mathcal{HS}_{EDF}$ algorithm is slightly adjusted for the revised problem. Compared to $\mathcal{HS}_{EDF}$ to Problem 5.1, there is little difference for the CPU utilization resulting from $\mathcal{HS}_{EDF}$ for both problems. This indicates that the relaxation of the deadline constraint has little impact on the resulted CPU utilization of transactions. It only slightly improves the set of sensor transactions that can be scheduled.

7 Conclusions

Consistency maintenance of data is an important problem in real-time applications. We have proposed three novel approaches, namely $\mathcal{ML}_{EDF}$, $\mathcal{OS}_{EDF}$ and $\mathcal{HS}_{EDF}$, using the *EDF* scheduling algorithm. $\mathcal{ML}_{EDF}$ is a linear algorithm but it only supports transactions with deadlines no greater than their corresponding periods (i.e., $D_i \leq P_i$ for τ_i). Our analysis for $\mathcal{ML}_{EDF}$ in Section 4.3 sheds light on how much $\mathcal{ML}_{EDF}$ can improve over existing *HH* approach quantitatively. The other two approaches outperform $\mathcal{ML}_{EDF}$ as they are derived from processor demand analysis (Lemma 4.1), a more accurate feasibility condition for *EDF* scheduling. This is clearly demonstrated in our experimental results. In contrast, $\mathcal{OS}_{EDF}$ and $\mathcal{HS}_{EDF}$ support transactions with arbitrary deadlines. In particular, $\mathcal{OS}_{EDF}$ is an algorithm that yields minimized processor utilization for periodic sensor transactions although it is not as efficient as the heuristic algorithm $\mathcal{HS}_{EDF}$. Our experimental results demonstrate that $\mathcal{HS}_{EDF}$ is an effective algorithm that assigns periods and deadlines with much lower utilization than $\mathcal{ML}_{EDF}$.

However, more investigation is necessary to understand the performance difference of the alternate approaches studied in this paper. In particular, we need to better understand how HS_{EDF} performs in comparison to EML_{DM}. For scheduling transactions with *arbitrary* deadlines, one of the important open questions is whether there is any sufficient and necessary condition for schedulability of *EDF* in temporal consistency maintenance. Further investigation on those questions will help shed light on existing approaches for temporal consistency maintenance.

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] S. K. Baruah, R. R. Howell, L. E. Rosier, "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor," *Real-Time Systems*, 2(4), pp. 301-324, 1990.

[3] 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.

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

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

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

[7] F. S. Hiller, G. J. Lieberman, "Introduction to Operations Research," *McGraw-Hill Publishing Company*, 1990.

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

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

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

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

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

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

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

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

[16] I. Ripoll, A. Crespo, and A. Mok, "Improvement in Feasibility Testing for Real-Time Tasks," *Real-Time Systems*, 11(1): 19-39, 1996.

[17] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On Task Schedulability in Real-Time Control Systems," *IEEE Real-Time Systems Symposium*, December 1996.

[18] D. Seto, J. P. Lehoczky, L. Sha, "Task Period Selection and Schedulability in Real-Time Systems," *IEEE Real-Time Systems Symposium*, December 1998.

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

[20] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, "Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms," Kluwer Academic Publishers, 1998.

[21] L. A. Wolsey, "Integer Programming," *John Wiley & Son*, New York, 1998.

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

[23] M. Xiong, K. Ramamritham, J. 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.

[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, B. Liang, K. Lam, and Y. Guo. "Quality of service guarantee for temporal consistency of real-time transactions," *IEEE Transactions on Knowledge and Data Engineering*, 18(8), pp. 1097-1110, 2006.

Appendix

Proof of Theorem 4.1: From the deadline constraint in Problem 4.2, we have

$$\begin{aligned} P_i \geq D_i \quad &\Longrightarrow \quad \frac{N_i - 1}{N_i}\mathcal{V}_i \geq \frac{1}{N_i}\mathcal{V}_i \Longrightarrow N_i \geq 2, \\ D_i \geq C_i \quad &\Longrightarrow \quad \frac{\mathcal{V}_i}{C_i} \geq N_i. \end{aligned}$$

Following Eq. 3 and 4, Problem 4.2 is reduced to the following non-linear programming problem with variable $\vec{N}$:

$$\min_{\vec{N}} \mathcal{U}, \text{ where } \mathcal{U} = \sum_{i=1}^{m} \frac{N_i C_i}{(N_i - 1)\mathcal{V}_i} \ (1 \leq i \leq m)$$

subject to:

$$N_i \geq 2 \tag{23}$$

$$\frac{\mathcal{V}_i}{C_i} \geq N_i \tag{24}$$

$$\sum_{i=1}^{m} N_i \cdot \frac{C_i}{\mathcal{V}_i} \leq 1 \tag{25}$$

It can be proved that the objective function and all three constraints are *convex* functions. Thus this is a convex programming problem, and a local minimum is a global minimum for this problem [7]. Considering Eq. 25 and Eq. 23 together, we have $2 \cdot \sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i} \leq \sum_{i=1}^{m} N_i \cdot \frac{C_i}{\mathcal{V}_i} \leq 1$. That is,

$$\sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i} \leq \frac{1}{2}. \tag{26}$$

Eq. 26 implies that $\gamma \leq \frac{1}{2}$.

For convenience, let $w_i = \frac{C_i}{\mathcal{V}_i}$, $\gamma = \sum_{i=1}^{m} \frac{C_i}{\mathcal{V}_i} = \sum_{i=1}^{m} w_i$, $x_i = N_i - 1$. By definition of $\mathcal{U}$ and Eq. 4, $\mathcal{U} = \sum_{i=1}^{m} \frac{N_i C_i}{(N_i - 1)\mathcal{V}_i}$ $= \sum_{i=1}^{m} \frac{x_i + 1}{x_i} w_i$, i.e., $\mathcal{U} = \sum_{i=1}^{m} w_i + \sum_{i=1}^{m} \frac{w_i}{x_i}$.

Following Eq. 23, 24 and 25, the problem is transformed to:

$$\min_{\vec{x}} \sum_{i=1}^{m} \frac{w_i}{x_i}$$

subject to:

$$x_i - 1 \geq 0 \tag{27}$$

$$\frac{1}{w_i} - x_i - 1 \geq 0 \tag{28}$$

$$1 - \gamma - \sum_{i=1}^{m} w_i \cdot x_i \geq 0 \tag{29}$$

Introducing Lagrangian multipliers $\lambda_{1,i}$, $\lambda_{2,i}$ and λ_3, we write Kuhn-Tucker condition as following $(1 \leq i \leq m)$:

$$-\frac{w_i}{x_i^2} + \lambda_3 w_i - \lambda_{2,i} + \lambda_{1,i} = 0, \tag{30}$$

$$\lambda_3 (1 - \gamma - \sum_{i=1}^{m} w_i x_i) = 0, \tag{31}$$

$$\lambda_3 \geq 0, \tag{32}$$

$$\lambda_{2,i}(x_i - 1) = 0, \tag{33}$$

$$\lambda_{2,i} \geq 0, \tag{34}$$

$$\lambda_{1,i}(\frac{1}{w_i} - x_i - 1) = 0, \tag{35}$$

$$\lambda_{1,i} \geq 0. \tag{36}$$

We use above conditions to construct an optimal solution. Suppose $\lambda_{1,i} = \lambda_{2,i} = 0$ and $\lambda_3 > 0$. Following Eq. 30, $-\frac{w_i}{x_i^2} + \lambda_3 w_i = 0$. Therefore, $x_i = \frac{1}{\sqrt{\lambda_3}}$ $(\lambda_3 > 0 \ \& \ 1 \leq i \leq m)$. Following $\lambda_3 > 0$ and Eq. 31, we have

$$1 - \gamma - \textstyle\sum_{i=1}^{m} w_i x_i = 0.$$

Replacing x_i with $\frac{1}{\sqrt{\lambda_3}}$,

$$1 - \gamma - \tfrac{1}{\sqrt{\lambda_3}} \textstyle\sum_{i=1}^{m} w_i = 0.$$

Replacing $\sum_{i=1}^{m} w_i$ with γ,

$$1 - \gamma - \frac{1}{\sqrt{\lambda_3}}\gamma = 0.$$

Solving the above equation, we have $\lambda_3 = (\frac{\gamma}{1-\gamma})^2$. It is easy to check that

$$\lambda_{1,i} = \lambda_{2,i} = 0,\ \lambda_3 = (\frac{\gamma}{1-\gamma})^2 \text{ and } x_i = \frac{1}{\sqrt{\lambda_3}} = \frac{1-\gamma}{\gamma}$$

satisfy Eq. 27 through Eq. 36, which means that $\vec{x}$ reaches a local minimum. Because the objective function is convex and constraints are all linear, $\vec{x}$ is also a global optimal solution. Since $N_i = x_i + 1$, $\mathcal{U}$ is minimized when $N_i = \mathcal{N}_{ml}^{opt} = \frac{1}{\gamma}$, and the minimum utilization is $\mathcal{U}_{ml}^{opt}(\gamma) = \frac{\mathcal{N}_{ml}^{opt}}{\mathcal{N}_{ml}^{opt}-1}\sum_{i=1}^{m}\frac{C_i}{\mathcal{V}_i} = \frac{\frac{1}{\gamma}}{\frac{1}{\gamma}-1}\gamma = \frac{\gamma}{1-\gamma}$. □

Proof of Theorem 4.2: Let $\mathcal{D}(\gamma) = \mathcal{U}_{hh}(\gamma) - \mathcal{U}_{ml}^{opt}(\gamma)$. From definitions of $\mathcal{U}_{ml}^{opt}(\gamma)$ and $\mathcal{U}_{hh}(\gamma)$, it follows that $\mathcal{D}(\gamma) = 2\gamma - \frac{\gamma}{1-\gamma}$. To obtain the maximum of $\mathcal{D}(\gamma)$, we differentiate $\mathcal{D}(\gamma)$ with respect to γ, and set the result to 0:

$\frac{d\mathcal{D}(\gamma)}{d\gamma} = \frac{2\gamma^2-4\gamma+1}{(1-\gamma)^2} = 0$

Thus the maximum of $\mathcal{D}(\gamma)$ is $3 - 2\sqrt{2}$ when $\gamma = 1 - \frac{\sqrt{2}}{2}$. □

Proof of Lemma 5.1: If $\mathcal{T}$ is not feasible, then $\mathcal{H}_{\mathcal{T}}(t) > t$ (Lemma 4.1). We need to find the maximal time $t = t_{\mathcal{B}}(\vec{P})$ so that $\mathcal{H}_{\mathcal{T}}(t) > t$ may hold in $[0, t_{\mathcal{B}}(\vec{P}))$. Because $P_i \geq C_i$, if

$$t \geq \max_i(\mathcal{V}_i - 2C_i), \tag{37}$$

then $(\lfloor\frac{(t-\mathcal{V}_i)}{P_i}\rfloor + 2) \geq 0$ (Remember that $P_i \geq C_i$). Suppose Eq. 37 holds, we have

$$\begin{aligned}
\mathcal{H}_{\mathcal{T}}(t) &= \textstyle\sum_{i=1}^{m} max(0, (\lfloor\frac{(t-\mathcal{V}_i)}{P_i}\rfloor + 2)\cdot C_i) \\
&= \textstyle\sum_{i=1}^{m}(\lfloor\frac{t-\mathcal{V}_i}{P_i}\rfloor + 2)\cdot C_i) \quad \{\text{Eliminating the max function}\} \\
&\leq \textstyle\sum_{i=1}^{m}\frac{C_i}{P_i}t + \sum_{i=1}^{m} C_i(2 - \frac{\mathcal{V}_i}{P_i}) \quad \{\text{Eliminating the floor function}\}
\end{aligned}$$

If

$$t\sum_{i=1}^{m}\frac{C_i}{P_i} + \sum_{i=1}^{m} C_i(2 - \frac{\mathcal{V}_i}{P_i}) \leq t, \tag{38}$$

then $\mathcal{H}_{\mathcal{T}}(t) \leq t$. Solving Eq. 38, we have

$$t \geq \frac{\sum_{i=1}^{m}(2 - \frac{\mathcal{V}_i}{P_i})C_i}{1 - \sum_{i=1}^{m}\frac{C_i}{P_i}}.$$

Considering Eq. 37, we have $\mathcal{H}_{\mathcal{T}}(t) \leq t$ if

$$t \geq max(\max_i(\mathcal{V}_i - 2C_i), \frac{\sum_{i=1}^{m}(2 - \frac{\mathcal{V}_i}{P_i})C_i}{1 - \mathcal{U}}).$$

Following Lemma 4.1, $\mathcal{T}$ is feasible iff $\forall t < t_{\mathcal{B}}(\vec{P}), \mathcal{H}_{\mathcal{T}}(t) \leq t$. □

Proof of Theorem 5.1: Given a solution $\mathcal{K}$ of Problem 5.2 with deadlines and periods derived from a scheduler $\mathcal{S}$, suppose that utilization of $\mathcal{K}$ is $\mathcal{U}_{\mathcal{K}}$, and $\mathcal{U}_{\mathcal{K}} < \mathcal{U}_{EDF}^{opt}$. $\mathcal{K}$ is feasible if it is scheduled by $\mathcal{S}$. Since *EDF* is an optimal scheduler [20],

if $\mathcal{K}$ can be scheduled by $\mathcal{S}$ then it can also be scheduled by *EDF*. Thus, $\mathcal{K}$ is also a feasible solution for Problem 5.1. But $\mathcal{U}_{\mathcal{K}} < \mathcal{U}_{EDF}^{opt}$ contradicts that $\mathcal{U}_{EDF}^{opt}$ is the optimized (minimized) utilization for Problem 5.1. Thus $\mathcal{U}_{EDF}^{opt}$ is optimal for Problem 5.2. □

Proof of Lemma 5.2: First, we prove that $\mathcal{U}^K \leq \mathcal{U}_{EDF}^{opt}$. Suppose $\vec{P}^*$ is the optimal solution to Problem 5.1, then P_i^* $(i = 1, ..., m)$ is an integer between C_i and $\mathcal{V}_i - C_i$ that can be expressed as

$$P_i^* = \sum_{j=C_i}^{\mathcal{V}_i - C_i} j z_{i,j}^*,$$

where $z_{i,j}^* = 1$ if $j = P_i^*$; otherwise, $z_{i,j}^* = 0$. Note that

$$\frac{1}{P_i^*} = \sum_{j=C_i}^{\mathcal{V}_i - C_i} \frac{z_{i,j}^*}{j}. \tag{39}$$

For any $k = 0, 1, .., K$, $z_{i,j}^*$ $(C_i \leq j \leq \mathcal{V}_i - C_i)$ (which determines $\vec{P}^*$) is thus a feasible solution to Problem 5.3 because (1) $\vec{P}^*$ satisfies Constraint 9; (2) the set of time constraints of Problem 5.3 (Constraint 10) is a *subset* of that of Problem 5.1 (Eq. 7). Following Eq. 10,

$\sum_{i=1}^{m} C_i \sum_{j=C_i}^{\mathcal{V}_i - C_i} z_{i,j}^* \max(0, \lfloor \frac{t_k - \mathcal{V}_i}{j} \rfloor + 2)$
$= \sum_{i=1}^{m} \max(0, (\lfloor \frac{t_k - \mathcal{V}_i}{P_i^*} \rfloor + 2) C_i)$
$\{$Moving C_i and $z_{i,j}^*$ into the max function, and by Eq. 39 $\}$
$\leq t_k$ $\{P_i^*$ satisfying Eq. 7. $\}$

By definition of Problem 5.3, all feasible solutions that satisfy Problem 5.3, $z_{i,j} = z_{i,j}^K$ produces the minimum $\mathcal{U}^K$. Therefore, $\mathcal{U}^K \leq \mathcal{U}_{EDF}^{opt}$.

We can prove that $\mathcal{U}^{n-1} \leq \mathcal{U}^n$ $(1 \leq n \leq K)$ similarly. □

Proof of Lemma 5.3: 1. **(If)** By the definition of F^K, if $F^K \geq 0$ then

$$t \geq \sum_{i=1}^{m} \max\{0, (\lfloor \frac{t - \mathcal{V}_i}{P_i^K} \rfloor + 2) C_i\}.$$

This implies that Eq. 7 holds. So $\vec{P}^K$ satisfies both the *period* and *time* constraints in Problem 5.1. By Lemma 5.2, $\mathcal{U}^K \leq \mathcal{U}_{EDF}^{opt}$. Thus $\vec{P}^K$ also minimizes processor utilization in Problem 5.1, and it is an optimal solution.

2. **(Only if)** This can be proved in a manner similar to the *if* case. □

Proof of Theorem 5.2: If $\vec{P}(t')$ is obtained after Step 4 in Alg. 5.2 at time $t' \geq 0$, then $\mathcal{H}_{\mathcal{T}(t')}(t') \leq t'$ for $\vec{P}(t')$'s corresponding transaction set $\mathcal{T}(t')$. First, $\vec{P}(0)$ must be feasible for the time constraint at $t' = 0$. Otherwise, the selection problem is unsolvable and the algorithm is terminated, which is contradictory to the assumption that $\vec{P}(t')$ is obtained after Step 4 at time $t' \geq 0$.

Suppose that $\vec{P}(t')$ satisfies the time constraint at all $t \leq t'$, i.e., $\mathcal{H}_{\mathcal{T}(t')}(t) \leq t$. If $\vec{P}(t')$ also satisfies $\mathcal{H}_{\mathcal{T}(t')}(t'+1) \leq t'+1$ at time $t'+1$, then set $\vec{P}(t'+1) = \vec{P}(t')$ and $\mathcal{H}_{\mathcal{T}(t')}(t) \leq t$ holds for $t \in [0, t'+1]$. Otherwise, $P_i(t')$ $(\tau_i \in \mathcal{R}(t'))$ is

reduced, and $\vec{P}(t')$ is changed to $\vec{P}(t'+1)$, which satisfies the time constraint at $t'+1$ (by Step 4). Furthermore, P_i is only changed for $\tau_i \in \mathcal{R}(t')$. By $\mathcal{R}(t')$ definition (Eq. 19), we have

$$P_i(t'+1) = P_i(t') \text{ if } t'+1 \geq \mathcal{V}_i, \tag{40}$$

$$P_i(t'+1) \leq P_i(t') \text{ if } t'+1 < \mathcal{V}_i. \tag{41}$$

Given $t < t'+1$ and the definition of $\mathcal{H}_{\mathcal{T}(t'+1)}(t)$,

$$\begin{aligned}
& \mathcal{H}_{\mathcal{T}(t'+1)}(t) \\
= & \sum_{i=1}^{m} max(0, (\lfloor \frac{(t-\mathcal{V}_i)}{P_i(t'+1)} \rfloor + 2) \cdot C_i) \\
\leq & \sum_{i=1}^{m} max(0, (\lfloor \frac{(t-\mathcal{V}_i)}{P_i(t')} \rfloor + 2) \cdot C_i) \ \{\text{By Eq. 40, 41, and } t-\mathcal{V}_i < 0 \text{ if } t'+1 < \mathcal{V}_i\} \\
= & \mathcal{H}_{\mathcal{T}(t')}(t) \\
\leq & t \ \{\text{By induction assumption}\}
\end{aligned}$$

Therefore, for all $t \leq t'+1$, $\mathcal{H}_{\mathcal{T}(t'+1)}(t) \leq t$, which proves the theorem. □

Proof of Lemma 5.4: Note that $J_{i,j}$ is guaranteed by *EDF* scheduling to complete by $d_{i,j}$ (note that $r_{i,j+1} < d_{i,j}$ if $D_i > P_i$). If it cannot be executed before time $r_{i,j+1}$, then it will have C_i time units allocated from the processor for its execution in $[r_{i,j+1}, d_{i,j}]$. Such C_i time units can also be used by $J_{i,j+1}$ if $J_{i,j}$ is skipped. □