This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

Temporal Consistency Maintenance Upon Partitioned Multiprocessor Platforms

Jianjun Li, Jian-Jia Chen, Ming Xiong, Guohui Li, and Wei Wei

Abstract—Maintaining timeliness and data freshness for real-time data objects has long been recognized as an important problem in real-time database research. Despite years of active research, most of the past work focuses on uniprocessor systems. In this paper, we study the workload-aware temporal consistency maintenance problem upon multiprocessor platforms. We consider the problem of how to partition a set of update transactions to $m \geq 2$ processors to maintain the temporal consistency of real-time data objects under both earliest deadline first (EDF) and deadline monotonic (DM) scheduling in each processor, while minimizing the total workload on m processors. Firstly, we only consider the feasibility aspect of the problem by proposing two polynomial time partitioning schemes, Temporal Consistency Partitioning under EDF (TCP_{EDF}) and Temporal Consistency Partitioning under DM (TCP_{DM}), and formally showing that the resource augmentation bounds of both TCP_{EDF} and TCP_{DM} are $(3 - \frac{1}{m})$. Secondly, we address the partition problem globally by proposing a polynomial time heuristic, Density factor Balancing Fit (DBF), where density factor balancing plays a major role in producing workload-efficient partitionings. Finally, we evaluate the feasibility and workload performances of DBF versus other heuristics with comparable quality experimentally.

Keywords—Real-Time Database; Temporal Consistency; Update Transaction; Multiprocessor; Partitioning Scheduling

1 Introduction

Real-Time database systems (RTDBS) have been widely used in many applications that require processing of massive amount of real-time data in a timely manner, such as aerospace and defense systems, industrial automation and air traffic control systems. Typically, a real-time database (RTDB) is composed of real-time objects which are updated by periodic sensor update transactions. An object in the database models the current status of a real world entity in the external environment, for example, the longitude, latitude and velocity of an aircraft. Different from data stored in traditional databases, the state of a real-time object may become invalid with the passage of time. Associated with the state is a temporal validity interval. To monitor the states of objects faithfully, a real-time object must be refreshed by a sensor update transaction before it becomes invalid, i.e., before its temporal validity interval expires, otherwise the RTDBS cannot respond to environmental changes timely.

The actual length of the temporal validity interval of a real-time object is usually application-dependent [28]. Sensor update transactions are generated by intelligent sensors, which are incorporated with dedicated signal processing functions and periodically sample the values of real-time objects. When sensor update transactions arrive at RTDBs with sampled data values, their updates are issued and real-time data are refreshed. Given the temporal consistency requirement, one important issue in designing RTDBS is to schedule sensor update transactions so that the temporal consistency of real-time data objects can be maintained while the resulting processor workload can be minimized. There are important reasons to minimize the processor workload imposed by sensor update transactions [16], [36], [37]: (1) it helps save sensor energy, because an inappropriate and unnecessarily short period of sensor update transactions may drive the sensor batteries flat quickly; (2) given the same portion of processor capacity, the RTDBS is able to accommodate more sensor update transactions; and (3) system efficiency can be improved because more processor capacity can be left to other user transactions that are triggered due to environmental changes brought by sensor update transactions.

In the past, while there has been much work devoted to the temporal consistency scheduling problem, most of them are focused on uniprocessor systems. Some examples are Half-Half (*HH*) [16], More-Less ($\mathcal{ML}_{DM}$) [37], *DS-FP* [13], [36], and $\mathcal{HS}_{EDF}$ [38]. In this work, we address the workload-aware temporal consistency maintenance problem on multiprocessor platforms, while in each single processor we consider both dynamic priority scheduling by the Earliest Deadline First (EDF) [27] algorithm and static priority scheduling by the Deadline Monotonic (DM) [24] algorithm. Our aim is to design efficient schemes which can allocate sensor update transactions to processors to guarantee the temporal consistency of real-time data objects, while, at the same time, reducing the workload (or utilization) of sensor update transactions on all the processors to the greatest extent.

- *Jianjun Li, Guohui Li and Wei Wei are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei, 430074, P.R.China. E-mail: jianjunli@hust.edu.cn, guohuili@hust.edu.cn, weiwei8329@gmail.com.*
- *Jian-Jia Chen is with the Department of Informatics, TU Dortmund University, D-44227 Dortmund, Germany. Email: jian-jia.chen@cs.uni-dortmund.de.*
- *Ming Xiong is with Google Inc. USA. Email: mxiong@google.com.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

Multiprocessor scheduling of periodic tasks is one of the most extensively studied areas in real-time systems research. In general, the approaches fall into either *global* or *partitioned* scheduling categories. In global scheduling [3], [4], [6], there is a single *ready queue* and task migrations among processors are allowed, i.e., each task can execute on any available processor in the run time. In contrast, partitioned scheduling [1], [5], [9] allocates each task to one processor permanently (task migrations are not allowed) and resorts to well-established single-processor scheduling schemes to guarantee feasibility. Global and partitioned approaches are known to have their own advantages and disadvantages in traditional multiprocessor real-time scheduling [11]. Recently, a number of works [2], [20] have been conducted on partitioned scheduling with task splitting (also referred to *semi-partitioned* scheduling). In this class of scheduling, while most tasks are statically assigned to a fixed processor as in partitioned scheduling, a small number of tasks are split into several subtasks, and each subtask is assigned and execute on a different but fixed processor. A recent survey on multiprocessor real-time scheduling can be found in [8].

This work: We consider the problem of workload minimization for periodic preemptive real-time sensor update transactions that are scheduled on an identical multiprocessor platform. We adopt partitioned scheduling and in each single processor, we consider both EDF and DM scheduling. Partitioning-based multiprocessor real-time scheduling considers *feasibility* as the main objective. The problem is invariably NP-Hard and appears in two variations: Minimizing the number of processors needed to guarantee the feasibility of the task set, or alternatively, given a *fixed* multiprocessor platform, finding sufficient schedulability (utilization) bounds. Our work opts for the second setting, thus we assume the existence of a given number of processors. The main contribution of this paper can be summarized as follows:

1) We first examine the feasibility aspect of the workload-aware transaction assignment problem by proposing a polynomial-time partition algorithm, Temporal Consistency Partitioning under EDF ($\mathsf{TCP}_{\mathsf{EDF}}$). We also formally show that the resource augmentation bound of $\mathsf{TCP}_{\mathsf{EDF}}$ is $(3 - \frac{1}{m})$.
2) We theoretically prove that the upper density factor[1] bound for temporal consistency maintenance under DM scheduling on a single processor is $\frac{1}{2}$. Based on this result, we propose a polynomial-time partition algorithm, Temporal Consistency Partitioning under DM ($\mathsf{TCP}_{\mathsf{DM}}$), and show that $\mathsf{TCP}_{\mathsf{DM}}$ has the same resource augmentation bound of $(3 - \frac{1}{m})$ as $\mathsf{TCP}_{\mathsf{EDF}}$.
3) We characterize the workload-efficient transaction-to-processor assignment problem as a density factor balance problem and propose a polynomial time heuristic, Density factor Balancing Fit (DBF).
4) We evaluate and comment on the performance of DBF via extensive simulation experiments. Our experimental study shows that DBF has better feasibility/workload performance than other heuristics with comparable quality.

1. Note here our definition of density factor, which will be detailed in Section 2, is different from the definition of the same words [8] in traditional multiprocessor real-time scheduling.

Organization: The remainder of this paper is organized as follows: Section 2 gives the definition of temporal validity and presents some notations and assumptions. The problem to be addressed is also introduced. In Section 3, we present and evaluate the performance of two polynomial-time partitioning schemes, $\mathsf{TCP}_{\mathsf{EDF}}$ and $\mathsf{TCP}_{\mathsf{DM}}$, for the partition problem when only considering the feasibility aspect. Section 4 addresses the problem globally by detailing the design of our heuristic DBF. Experimental evaluation results of DBF versus other heuristics are described in Section 5. Section 6 briefly reviews some related work and finally, conclusions are drawn in Section 7.

2 Background, Assumptions and Problem Definition

In this section, we first review the definition of temporal validity for data freshness, and then present some notations as well as important assumptions made throughout the paper. We also briefly introduce $\mathcal{GE}_{EDF}$ and $\mathcal{ML}_{DM}$, which will be used as the period and deadline calculation schemes on each single processor under EDF and DM scheduling, respectively. Finally, we define the problem to be addressed in this work.

2.1 Temporal Validity for Data Freshness

In an RTDB, a data object is a logical image of a real-world entity. As the state of a real-world entity changes continuously, to monitor the entity's state faithfully, real-time data objects must be refreshed by update transactions, which are generated periodically by intelligent sensors, before they become invalid. The actual length of the temporal validity interval of a real-time data object is usually application dependent [28], [31].

Definition 1. *[31] A real-time data object (x_i) at time t is temporally valid if, for its j^{th} update finished last 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., $r_{i,j} + \mathcal{V}_i \geq t$.*

According to Definition 1, a value for real-time data object x_i sampled at any time t will be valid from t up to $(t + \mathcal{V}_i)$. To satisfy the validity constraint, for each x_i, suppose τ_i is the sensor transaction that is responsible for updating x_i, then the farthest distance (based on the arrival time of an instance of τ_i and the finishing time of its next instance) of two consecutive sensor transactions of τ_i is $\mathcal{V}_i$, which means τ_i should execute at least twice during $\mathcal{V}_i$. Traditional methods, such as *Half-Half*, *More-Less* (for both EDF and DM) and *DS-FP*, have been proposed to solve the problem of how to assign periods

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

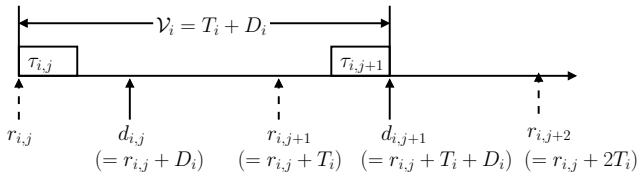

Fig. 1: Illustration of *More-Less* scheme.

and deadlines to update transactions and schedule them to maintain the validity consistency while minimizing the utilization on a single processor. Among these approaches, *More-Less* [37] is the best one based on periodic transaction model for both EDF and DM scheduling. Illustration of *More-Less* is depicted in Figure 1. As can be observed, in order to satisfy the validity constraint (execute twice during $\mathcal{V}_i$) and minimize the workload, there is $\mathcal{V}_i = T_i + D_i$, where T_i and D_i represent the period and deadline of τ_i, respectively. Hence, in the following discussion, if not explicitly indicated, we assume $\mathcal{V}_i = T_i + D_i$.

2.2 Notations and Assumptions

In this paper, we use $\mathcal{T} = \{\tau_i\}_{i=1}^n$ and $\mathcal{X} = \{x_i\}_{i=1}^n$ to denote a set of periodic sensor update transactions and a set of real-time or temporal data, respectively. We consider the scheduling of $\mathcal{T}$ on a set of m-*identical* multiprocessors $\mathcal{M} = \{M_i\}_{i=1}^m$ and adopt a partitioning-based approach to multiprocessor scheduling. Transactions are assigned *permanently* to processors. On each processor, determined by whether the EDF or DM scheduling algorithm is adopted, we use $\mathcal{GE}_{EDF}$ or $\mathcal{ML}_{DM}$, which will be introduced later, to derive period and deadline for transactions. All temporal data are assumed to be kept in main memory. Each data x_i $(1 \leq i \leq n)$ is associated with a validity interval length $\mathcal{V}_i$. Transaction τ_i is responsible for updating the corresponding data x_i periodically. Since each sensor transaction updates different data, no concurrency control is considered. Each update transaction τ_i is periodic and is characterized by the following 3-tuple: $\{C_i, D_i, T_i\}$, where C_i is the execution time, D_i is the relative deadline and T_i is the period. Transaction deadlines are considered to be *constrained*, i.e., $D_i \leq T_i$. We use U_i and λ_i to denote the utilization and density factor of τ_i, respectively, i.e., $U_i = \frac{C_i}{T_i}$ and $\lambda_i = \frac{C_i}{\mathcal{V}_i}$. Since each transaction must be assigned to exactly one processor, it is clear that the total utilization and total density factor of $\mathcal{T}$ are $U_{sum} = \sum_{i=1}^n \frac{C_i}{T_i}$ and $\lambda_{sum} = \sum_{i=1}^n \frac{C_i}{\mathcal{V}_i}$, respectively. When there is no confusion, we also use $U_{\mathcal{T}}$ ($\lambda_{\mathcal{T}}$, resp.) to denote the workload (density factor, resp.) of $\mathcal{T}$. Lastly, we use $\mathcal{T}(M_k)$ to denote the transactions that have been assigned to M_k. Formal definitions of symbols used in this paper are presented in Table 1.

2.3 $\mathcal{GE}_{EDF}$: Deadline and Period Calculation Scheme under EDF scheduling

To derive periods and deadlines which can guarantee temporal validity of data objects while minimizing the workload of update transactions scheduled under EDF on a single processor, $\mathcal{OS}_{EDF}$ and $\mathcal{HS}_{EDF}$ are proposed in [38]. $\mathcal{OS}_{EDF}$ is a *branch-and-bound*-based search algorithm which can find the optimal solutions. The problem with $\mathcal{OS}_{EDF}$ is that it does not scale well with increasing problem size. $\mathcal{HS}_{EDF}$ is a search-based heuristic and is capable of finding a solution if one exists. Compared with $\mathcal{OS}_{EDF}$, $\mathcal{HS}_{EDF}$ is more efficient, and its efficiency is achieved at the expense of increased processor workload. To further advance the state-of-the-art, the $\mathcal{GE}_{EDF}$ algorithm is proposed in [26]. $\mathcal{GE}_{EDF}$ is a two-phase algorithm. The first phase can find a solution in linear time, while the second phase is a search scheme based on the initial result of $\mathcal{ML}_{DM}$, an algorithm which will be introduced next. Compared to $\mathcal{HS}_{EDF}$, $\mathcal{GE}_{EDF}$ is able to find a solution with lower (or equal at most) workload in a more time-efficient manner. Since computing deadline and period for transactions under EDF scheduling is not the focus of this paper, we do not present the detail and would like to refer readers to [38] and [26] for full descriptions of the above three mentioned algorithms.

TABLE 1: Symbols and definitions

Symbol	Definition
x_i	Real-time data object i
τ_i	Sensor update transaction updating x_i
C_i	Execution time of τ_i
$\mathcal{V}_i$	Validity interval length of x_i
T_i	Period of τ_i
D_i	Relative deadline of τ_i
U_i	Processor workload (or utilization) of τ_i
U_{sum}	Workload of $\{\tau_i\}_{i=1}^n$
$U_{\mathcal{T}}$	Workload of $\mathcal{T}$
λ_i	Density factor of τ_i, $\lambda_i = \frac{C_i}{\mathcal{V}_i}$
$\lambda_{\max}$	Maximum density factor among $\tau_1, \tau_2, \ldots, \tau_n$
λ_{sum}	Density factor of $\{\tau_i\}_{i=1}^n$
$\lambda_{\mathcal{T}}$	Density factor of $\mathcal{T}$
$\mathcal{M}$	A set of multiprocessors $\{M_1, M_2, \ldots M_m\}$
$\mathcal{T}(M_k)$	Transactions which have been assigned to M_k

2.4 $\mathcal{ML}_{DM}$: Deadline and Period Calculation Scheme under DM scheduling

To guarantee temporal validity of data objects while minimizing the workload of update transactions scheduled under DM on a single processor, $\mathcal{ML}_{DM}$ is proposed in [37]. $\mathcal{ML}_{DM}$ works as follows: Given a transaction set indexed in *Shortest Validity First* (SVF) order, i.e., $\mathcal{V}_i \leq \mathcal{V}_{i+1}$ $(i = 1, \ldots, n-1)$, $\mathcal{ML}_{DM}$ first computes deadline D_i for τ_i by finding the minimum solution of the recursive equation (starting with $D_i = C_i$)

$$\sum_{j=1}^{i} \lceil D_i/T_j \rceil C_j = D_i \tag{1}$$

and then assigns period to τ_i by $T_i = \mathcal{V}_i - D_i$. Note here the SVF order means $\mathcal{ML}_{DM}$ starts the computation from the update transaction with the shortest validity interval. It should be pointed out that although $\mathcal{ML}_{DM}$ has been reported to exhibit higher processor workload than EDF-based deadline and period calculation schemes such as $\mathcal{HS}_{EDF}$ and $\mathcal{GE}_{EDF}$, it is still the best choice under fixed-priority scheduling.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

2.5 Problem Definition

Our aim in this research effort is to address the following workload-aware real-time temporal consistency scheduling problem (denoted by W-PARTITION).

W-PARTITION: Given a set $\mathcal{T}$ of real-time update transactions and a set $\mathcal{M}$ of m identical processors, find a transaction-to-processor assignment and compute deadline and period for transactions on each single processor such that:

1. *the transactions assigned to each processor can be scheduled under EDF or DM in a feasible manner and,*
2. *the total workload of $\mathcal{M}$ is minimized (among all feasible transaction allocations).*

It can be observed that W-PARTITION is NP-Hard in the strong sense. Given a set of tasks with known execution times and with the same relative deadline/period as T on m identical processors, determining a feasible task assignment to meet the timing constraint is NP-Complete in the strong sense [30]. The reduction is as follows: For each task, we generate a corresponding update transaction by setting its execution time as the execution time of the task and the validity interval length as $2T$. Clearly, there exists a feasible solution for the reduced W-PARTITION input instance if and only if the input task set has a feasible solution. Therefore, deriving a feasible solution for W-PARTITION is NP-Complete in the strong sense. With the optimization of total workload, W-PARTITION is NP-Hard in the strong sense.

3 Temporal Consistency Partition

In view of the intractability of the problem, we first do not take the workload issue into consideration, but only focus on how to derive feasible partitionings. Specifically, for the multiprocessor platforms where transactions are scheduled under EDF in each single processor, we propose a polynomial-time partition algorithm: *Temporal Consistency Partition* under EDF ($\mathsf{TCP}_{\mathsf{EDF}}$), and offer the theoretical evaluation of it in Section 3.1. Correspondingly, for the platforms where transactions are scheduled under DM in each single processor, we present a polynomial-time partition algorithm: *Temporal Consistency Partition* under DM ($\mathsf{TCP}_{\mathsf{DM}}$), and provide the theoretical evaluation of it in Section 3.2. The discussion of addressing the whole W-PARTITION problem is left in Section 4.

3.1 $\mathsf{TCP}_{\mathsf{EDF}}$: Temporal Consistency Partitioning under EDF

3.1.1 Design of TCP_{EDF}

First, to distinguish from the traditional real-time multiprocessor scheduling problem, we make the following definition.

Definition 2. *A transaction set $\mathcal{T}$ is said to be temporal consistency schedulable under EDF with constrained deadlines if, for each transaction, a two-tuple of period and deadline can be derived by a period and deadline calculation algorithm (e.g., $\mathcal{HS}_{EDF}$, $\mathcal{GE}_{EDF}$) to make $\mathcal{T}$ schedulable under EDF with constrained deadlines on a single processor.*

Below we present a useful theorem, which identifies a sufficient condition for any transaction set to be temporal consistency schedulable under EDF on uniprocessor systems, and thus paves the way for our design of $\mathsf{TCP}_{\mathsf{EDF}}$.

Theorem 1. *Given a transaction set $\mathcal{T}$, if the density factor of $\mathcal{T}$ is not larger than 0.5, i.e., $\lambda_{sum} \leq 0.5$, then $\mathcal{T}$ is temporal consistency schedulable under EDF on a uniprocessor system.*

Proof: Given that $\lambda_{sum} \leq 0.5$, it is obvious for each transaction τ_i, we can derive a two-tuple of period and deadline by setting both T_i and D_i to be half of τ_i's validity interval length $\frac{\mathcal{V}_i}{2}$. We then have,

$$U_{sum} = \sum_{i=1}^{n} \frac{C_i}{T_i} = \sum_{i=1}^{n} \frac{C_i}{\mathcal{V}_i/2} = 2\lambda_{sum} \leq 1, \tag{2}$$

which means $\mathcal{T}$ is EDF-schedulable, and further, is temporal consistency schedulable under EDF, the theorem thus follows. □

Based on Theorem 1, we design the $\mathsf{TCP}_{\mathsf{EDF}}$ algorithm as follows: For any processor M_k, let $\mathcal{T}(M_k)$ denote the transactions from among $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ that have already been allocated to processor M_k. Considering the processors $M_1, M_2, \ldots, M_m$, in any order, Algorithm $\mathsf{TCP}_{\mathsf{EDF}}$ assigns transaction τ_i to a processor M_k, that satisfies the following condition:

$$\lambda_i + \sum\nolimits_{\tau_j \in \mathcal{T}(M_k)} \lambda_j \leq \frac{1}{2}. \tag{3}$$

If no such M_k exists, then $\mathsf{TCP}_{\mathsf{EDF}}$ declares failure: it is unable to conclude that the transaction set $\mathcal{T}$ is feasible upon the m-processor platform.

Since $\lambda_{sum} \leq 0.5$ is a sufficient condition for any transaction set to be temporal consistency schedulable under EDF on a single processor, it is straightforward to conclude that $\mathsf{TCP}_{\mathsf{EDF}}$ can guarantee the assignment is feasible if it succeeds to return a partition on transaction set $\mathcal{T}$. $\mathsf{TCP}_{\mathsf{EDF}}$ is quite time-efficient due to the reason that when allocating one transaction τ_i, it only needs to evaluate the density factor of the previously allocated $(i-1)$ transactions on each of the m processors by (3). Since this value can be computed in constant time, it is obvious that the run-time of $\mathsf{TCP}_{\mathsf{EDF}}$ in assigning all n transactions is no more than $\mathcal{O}(nm)$. Next, we offer a quantitative evaluation of the efficacy of Algorithm $\mathsf{TCP}_{\mathsf{EDF}}$.

3.1.2 Theoretical Evaluation of TCP_{EDF}

The technique of resource augmentation has been widely used to quantify the "goodness" of an algorithm for solving problems for which optimal solutions are either computationally intractable or just impossible in practice. In this technique, the performance of a given algorithm is compared with that of a hypothetical optimal one, under the assumption that the given algorithm can access more

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

resources (e.g., more processors, or processors of higher speeds) than the optimal algorithm. The partitioned multiprocessor real-time scheduling for sporadic real-time tasks, in which the relative deadlines are different from the periods, has been recently studied by Baruah and Fisher in [5], Chen and Chakraborty in [7], and Fisher et al. in [9]. Resource augmentation bounds have been derived to quantify the worst-case performance of their partition schemes. In this work, similar to [5], [7], [9], we also offer a quantitative evaluation of our algorithm in terms of resource augmentation bound. But it should be noted that the problem we addressed here is different from theirs in two ways: (1) transaction period and deadline are initially unknown in our problem; (2) the sum of a transaction's period and deadline is bounded by the validity interval length of the data object it updates, i.e., $T_i + D_i = \mathcal{V}_i$.

Below, we derive a resource augmentation bound (upper bound here) for $\mathsf{TCP}_{\mathsf{EDF}}$, which characterizes its performance. We first present a useful lemma.

Lemma 1. *If $\mathcal{T}$ is temporal consistency schedulable under EDF on an identical multiprocessor platform comprised of m processors each of computing capacity ξ, then we have that*

$$\lambda_{max} \leq \frac{1}{2} \cdot \xi \quad and \quad \lambda_{sum} < m \cdot \xi$$

Proof: Since each job of τ_i can receive at most $D_i \cdot \xi$ units of execution by its deadline, we have $C_i \leq D_i \cdot \xi \leq T_i \cdot \xi$. Given $\mathcal{V}_i = T_i + D_i$, we can get

$$\lambda_i = \frac{C_i}{\mathcal{V}_i} = \frac{C_i}{T_i + D_i} \leq \frac{C_i}{2D_i} \leq \frac{1}{2} \cdot \xi \tag{4}$$

Hence, $\lambda_{max} \leq \frac{1}{2} \cdot \xi$ indicates that no individual transaction's density factor may exceed half of the computing capacity of a processor.

For any transaction set on a uniprocessor, if it is temporal consistency schedulable under EDF, then its density factor should be less than its utilization, which in turn is 1 in the maximum. $\lambda_{sum} < m \cdot \xi$ thus reflects the requirement of the cumulative density factor on m processors of computing capacity ξ each. □

Note that Lemma 1 above essentially specifies a necessary condition for $\mathsf{TCP}_{\mathsf{EDF}}$ (or in fact, any partition algorithm) to successfully partition a transaction set. We now present a theorem below, which specifies a sufficient condition for $\mathsf{TCP}_{\mathsf{EDF}}$ to successfully partition a transaction set.

Theorem 2. *Any transaction set for which the following formula is true*

$$m \geq \frac{2\lambda_{sum} - 2\lambda_{max}}{1 - 2\lambda_{max}} \tag{5}$$

can be scheduled by $\mathsf{TCP}_{\mathsf{EDF}}$ on m unit-capacity processors.

Proof: We prove this by considering the case when $\mathsf{TCP}_{\mathsf{EDF}}$ fails to assign τ_i. In such a case, we know that on each processor M_k $(1 \leq k \leq m)$, there is

$$\lambda_i + \sum\nolimits_{\tau_j \in \mathcal{T}(M_k)} \lambda_j > \frac{1}{2} \tag{6}$$

Let $\mathcal{T}(\mathcal{M})$ denote $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ that have already been allocated to the m processors. By summing the above inequality on m processors, we get

$$m\lambda_i + \sum\nolimits_{\tau_j \in \mathcal{T}(\mathcal{M})} \lambda_j > \frac{1}{2} \cdot m \tag{7}$$

Since $\lambda_{sum} \geq \sum_{\tau_j \in \mathcal{T}(\mathcal{M})} \lambda_j + \lambda_i$, it is clear that

$$(m-1)\lambda_i + \lambda_{sum} > \frac{1}{2} \cdot m \;\Rightarrow\; m < \frac{2\lambda_{sum} - 2\lambda_i}{1 - 2\lambda_i} \tag{8}$$

Therefore, when

$$m \geq \frac{2\lambda_{sum} - 2\lambda_i}{1 - 2\lambda_i} \geq \frac{2\lambda_{sum} - 2\lambda_{max}}{1 - 2\lambda_{max}},$$

$\mathsf{TCP}_{\mathsf{EDF}}$ can successfully schedule $\mathcal{T}$ on m processors. □

By Theorem 2, we now present a resource augmentation result regarding $\mathsf{TCP}_{\mathsf{EDF}}$.

Theorem 3. *$\mathsf{TCP}_{\mathsf{EDF}}$ can guarantee the following performance: if a transaction set is temporal consistency schedulable under EDF on m identical processors each of computing capacity ξ, then $\mathsf{TCP}_{\mathsf{EDF}}$ can partition this transaction set on m processors that are each $\left(3 - \frac{1}{m}\right)$ times as fast as the original.*

Proof: Assume that $\mathcal{T} = \{\tau_1, \tau_2, \ldots \tau_n\}$ is a transaction set that is temporal consistency schedulable under EDF on m processors each of computing capacity ξ. We will prove below that $\mathcal{T}$ is guaranteed to be successfully partitioned by $\mathsf{TCP}_{\mathsf{EDF}}$ on m unit-capacity processors for $\xi \leq \frac{m}{3m-1}$.

Since $\mathcal{T}$ is temporal consistency schedulable under EDF on m ξ-speed processors, by Lemma 1, the transactions in $\mathcal{T}$ should satisfy the following properties:

$$\lambda_{max} \leq \frac{1}{2} \cdot \xi, \quad \lambda_{sum} < m \cdot \xi$$

By replacing the above conditions in inequality (5), we have

$$m \geq \frac{2\lambda_{sum} - 2\lambda_{max}}{1 - 2\lambda_{max}} \Leftarrow m \geq \frac{2m\xi - \xi}{1 - \xi}$$
$$\equiv \xi \leq \frac{m}{3m-1} \;\equiv\; \frac{1}{\xi} \geq 3 - \frac{1}{m},$$

which is as claimed in the theorem. □

3.2 $\mathsf{TCP}_{\mathsf{DM}}$: Temporal Consistency Partitioning under DM

3.2.1 Design of $\mathsf{TCP}_{\mathsf{DM}}$

Similar to the EDF case, we first make the following definition.

Definition 3. *A transaction set $\mathcal{T}$ is said to be* temporal consistency schedulable *under DM if, for each transaction, a two-tuple of period and deadline can be derived by a period and deadline calculation algorithm (e.g., $\mathcal{ML}_{DM}$) to make $\mathcal{T}$*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

TABLE 2: Transaction sets $\mathcal{T}$ and $\overline{\mathcal{T}}$

(a) Parameters of transactions in $\mathcal{T}$

WCET	Validity Interval	Deadline	Period
C_1	$\mathcal{V}_1$	D_1	T_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
C_{a-1}	$\mathcal{V}_{a-1}$	D_{a-1}	T_{a-1}
C_a	$\mathcal{V}_a$	D_a	T_a
C_{a+1}	$\mathcal{V}_{a+1}$	D_{a+1}	T_{a+1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
C_b	$\mathcal{V}_b$	D_b	T_b
$\vdots$	$\vdots$	$\vdots$	$\vdots$
C_{n-1}	$\mathcal{V}_{n-1}$	D_{n-1}	T_{n-1}
C_n	$\mathcal{V}_n$	D_n	T_n

(b) Parameters of transactions in $\overline{\mathcal{T}}$

WCET	Validity Interval	Deadline	Period
$\overline{C}_1 = C_1$	$\overline{\mathcal{V}}_1 = \mathcal{V}_1$	$\overline{D}_1 = D_1$	$\overline{T}_1 = T_1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\overline{C}_{a-1} = C_{a-1}$	$\overline{\mathcal{V}}_{a-1} = \mathcal{V}_{a-1}$	$\overline{D}_{a-1} = D_{a-1}$	$\overline{T}_{a-1} = T_{a-1}$
$\overline{C}_a = C_a$	$\overline{\mathcal{V}}_a = \mathcal{V}_a + \delta$	$\overline{D}_a = D_a$	$\overline{T}_a = T_a + \delta$
$\overline{C}_{a+1} = C_{a+1}$	$\overline{\mathcal{V}}_{a+1} = \mathcal{V}_{a+1}$	$\overline{D}_{a+1} = D_{a+1}$	$\overline{T}_{a+1} = T_{a+1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\overline{C}_b = C_b$	$\overline{\mathcal{V}}_b = \mathcal{V}_b$	$\overline{D}_b = D_b$	$\overline{T}_b = T_b$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\overline{C}_{n-1} = C_{n-1}$	$\overline{\mathcal{V}}_{n-1} = \mathcal{V}_{n-1}$	$\overline{D}_{n-1} = D_{n-1}$	$\overline{T}_{n-1} = T_{n-1}$
$\overline{C}_n = C_n$	$\overline{\mathcal{V}}_n = \mathcal{V}_n$	$\overline{D}_n = D_n$	$\overline{T}_n = T_n$

schedulable under DM with constrained deadlines on a single processor.

Below, we present a useful theorem (Theorem 4), which essentially identifies a sufficient condition for any transaction set to be temporal consistency schedulable under DM on uniprocessor systems. But first, we need an important lemma (Lemma 3), which will be used in the proof of Theorem 4. We start by giving two definitions as follows. Note in the remainder of this section, unless stated explicitly, we assume that the transactions in a transaction set are indexed in *Shortest Validity First* (SVF) order, i.e., $\forall i = 1, \ldots, n-1$, $\mathcal{V}_i \leq \mathcal{V}_{i+1}$.

Definition 4. *A transaction set $\mathcal{T} = \{\tau_i\}_{i=1}^n$ is said to be $\mathcal{ML}_{DM}$-schedulable if the deadline and period of each transaction τ_i can be derived by solving $D_i = \sum_{j=1}^{i-1} \lceil D_i/T_j \rceil C_j + C_i$ and $T_i = \mathcal{V}_i - D_i$, respectively, and $D_i \leq T_i$.*

Definition 5. *$\mathcal{T} = \{\tau_i\}_{i=1}^n$ is an* extreme *transaction set if $\mathcal{T}$ is $\mathcal{ML}_{DM}$-schedulable with $D_n = T_n$, and the density factor of $\mathcal{T}$, i.e., $\lambda_{\mathcal{T}}$, is smaller than or equal to the density factor of all other transaction sets consisting of n transactions that are $\mathcal{ML}_{DM}$-schedulable with $D_n = T_n$.*

From the above definition, we have the following property regarding *extreme* transaction set.

Lemma 2. *If $\mathcal{T} = \{\tau_i\}_{i=1}^n$ is an* extreme *transaction set, then for any two transactions τ_a and τ_b $(1 \leq a < b \leq n)$ in $\mathcal{T}$, there is,*

$$\left\lceil \frac{D_b}{T_a} \right\rceil > \frac{D_b}{T_a}$$

Proof: We prove it by contradiction. Since $\lceil D_b/T_a \rceil \geq D_b/T_a$ is always true, the opposition to $\lceil D_b/T_a \rceil > D_b/T_a$ is $\lceil D_b/T_a \rceil = D_b/T_a$. Without loss of generality, suppose transactions τ_a and τ_b $(1 \leq a < b \leq n)$ are the first two transactions which satisfy $\lceil D_b/T_a \rceil = D_b/T_a$. Now we have two cases to consider: (1) τ_a and τ_b $(1 \leq a < b \leq n)$ are the only pair of transactions that satisfies $\lceil D_b/T_a \rceil = D_b/T_a$, and for any other two transactions τ_c and τ_d $(1 \leq c < d \leq n)$, there is $\lceil D_d/T_c \rceil > D_d/T_c$; (2) Except for τ_a and τ_b, there exists some other pairs of transactions τ_c and τ_d $(a \leq c < d \leq n)$ that satisfies $\lceil D_d/T_c \rceil = D_d/T_c$.

We first consider Case (1). Based on $\mathcal{T}$, we construct a new transaction set $\overline{\mathcal{T}}$. In order to facilitate the distinction, we use $\overline{C}_i$ and $\overline{\mathcal{V}}_i$ to represent the execution time and validity interval length of transaction $\overline{\tau}_i$ in $\overline{\mathcal{T}}$, and use $\overline{D}_i$ and $\overline{T}_i$ to denote the corresponding deadline and period of $\overline{\tau}_i$.

We start by setting $\overline{C}_i = C_i$, $\overline{D}_i = D_i$, $\overline{T}_i = T_i$, and $\overline{\mathcal{V}}_i = \mathcal{V}_i$ for the first $(a-1)$ transactions. In other words, the first $(a-1)$ transactions $\overline{\tau}_i$ $(1 \leq i \leq a-1)$ remain the same as τ_i $(1 \leq i \leq a-1)$ in $\mathcal{T}$. Clearly, the first $(a-1)$ transactions in $\overline{\mathcal{T}}$ are still $\mathcal{ML}_{DM}$ schedulable. Now for transaction $\overline{\tau}_a$, we let $\overline{C}_a = C_a$ and $\overline{T}_a = T_a + \delta$, where δ is a sufficient small positive number, and use $\mathcal{ML}_{DM}$ to compute $\overline{D}_a$. Since the first $(a-1)$ transactions are the same as their correspondence in $\mathcal{T}$ ($\overline{C}_i = C_i$ and $\overline{T}_i = T_i$ for $1 \leq i \leq a-1$) and $\overline{C}_a = C_a$, from the computation process of $\mathcal{ML}_{DM}$, i.e., finding the minimum solution of the recursive equation $\sum_{j=1}^{a-1} \lceil \overline{D}_a/\overline{T}_j \rceil \overline{C}_j + \overline{C}_a = \overline{D}_a$, we know the calculated $\overline{D}_a$ is equal to D_a. Consequently, we have $\overline{\mathcal{V}}_a = \overline{T}_a + \overline{D}_a = \mathcal{V}_a + \delta$, as shown in Table 2(b).

We go on by considering the remaining transactions $\overline{\tau}_i$ $(a+1 \leq i \leq n)$. For $\overline{\tau}_{a+1}$, we let $\overline{C}_{a+1} = C_{a+1}$ and use $\mathcal{ML}_{DM}$ to calculate $\overline{D}_{a+1}$. Given that $\left\lceil \frac{D_{a+1}}{T_a} \right\rceil > \frac{D_{a+1}}{T_a}$, as long as δ is small enough such that $\frac{D_{a+1}}{T_a+\delta}$ is not an integer, we have $\left\lceil \frac{D_{a+1}}{\overline{T}_a} \right\rceil = \left\lceil \frac{D_{a+1}}{T_a+\delta} \right\rceil = \left\lceil \frac{D_{a+1}}{T_a} \right\rceil$. Moreover, since $\overline{C}_i = C_i$ $(1 \leq i \leq a+1)$ and $\overline{T}_i = T_i$ $(1 \leq i \leq a-1)$, by finding the minimum solution of

$$\sum\nolimits_{j=1}^{a} \left\lceil \frac{\overline{D}_{a+1}}{\overline{T}_j} \right\rceil \overline{C}_j + \overline{C}_{a+1} = \overline{D}_{a+1} \tag{9}$$

we can get that $\overline{D}_{a+1} = D_{a+1}$. Then by setting $\overline{T}_{a+1} = T_{a+1}$, we have $\overline{\mathcal{V}}_{a+1} = \mathcal{V}_{a+1}$. The following transactions $\overline{\tau}_i$ $(a+1 < i < b)$ can be processed in a similar way by first setting $\overline{C}_i = C_i$, $\overline{T}_i = T_i$ $(a+1 < i < b)$, and then calculating a deadline $\overline{D}_i$ $(a+1 < i < b)$ which is equal to D_i.

Now we proceed to transaction $\overline{\tau}_b$. First, we let $\overline{C}_b = C_b$ and $\overline{T}_b = T_b$. Given that $\left\lceil \frac{D_b}{T_a} \right\rceil = \frac{D_b}{T_a}$ and δ is a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

TABLE 3: Parameters of transaction set $\mathcal{T}'$

WCET	Validity Interval	Deadline	Period
$C'_1 = C_1 + \varepsilon_1$	$\mathcal{V}'_1 = \mathcal{V}_1$	$D'_1 = D_1 + \varepsilon_1$	$T'_1 = T_1 - \varepsilon_1$
$C'_2 = C_2 + \varepsilon_2$	$\mathcal{V}'_2 = \mathcal{V}_2$	$D'_2 = D_2 + k_2^1\varepsilon_1 + \varepsilon_2$	$T'_2 = T_2 - k_2^1\varepsilon_1 - \varepsilon_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$C'_i = C_i + \varepsilon_i$	$\mathcal{V}'_i = \mathcal{V}_i$	$D'_i = D_i + \sum_{j=1}^{i-1} k_i^j\varepsilon_j + \varepsilon_i$	$T'_i = T_i - \sum_{j=1}^{i-1} k_i^j\varepsilon_j - \varepsilon_i$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$C'_{n-1} = C_{n-1} + \varepsilon_{n-1}$	$\mathcal{V}'_{n-1} = \mathcal{V}_{n-1}$	$D'_{n-1} = D_{n-1} + \sum_{j=1}^{n-2} k_{n-1}^j\varepsilon_j + \varepsilon_{n-1}$	$T'_{n-1} = T_{n-1} - \sum_{j=1}^{n-2} k_{n-1}^j\varepsilon_j - \varepsilon_{n-1}$
$C'_n = C_n - \varepsilon_n$	$\mathcal{V}'_n = D'_n + T'_n$	$D'_n \leq D_n$	$T'_n = D'_n$

sufficiently small positive number, we can get $\left\lceil \frac{D_b}{T_a+\delta} \right\rceil = \left\lceil \frac{D_b}{T_a} \right\rceil$. Then, by finding the minimum solution of

$$\sum_{j=1}^{b-1} \left\lceil \frac{\overline{D}_b}{\overline{T}_j} \right\rceil \overline{C}_j + \overline{C}_b = \overline{D}_b \tag{10}$$

we can derive $\overline{D}_b = D_b$, which indicates that $\overline{\mathcal{V}}_b = \mathcal{V}_b$. The remaining transactions $\overline{\tau}_i$ $(b < i \leq n)$ can be handled in a similar way as transactions $\overline{\tau}_j$ $(a+1 < j < b)$ and we omit to detail the process here.

Up to now, we have constructed a new $\mathcal{ML}_{DM}$ schedulable transaction set $\overline{\mathcal{T}}$ with $\overline{D}_n = \overline{T}_n$, as shown in Table 2(b). Based on the parameters presented in Table 2, it can be verified that the density factor of $\overline{\mathcal{T}}$, $\lambda_{\overline{\mathcal{T}}} = \sum_{i=1}^{n} \frac{\overline{C}_i}{\overline{\mathcal{V}}_i}$, is strictly lower than the density factor of $\mathcal{T}$, $\lambda_{\mathcal{T}} = \sum_{i=1}^{n} \frac{C_i}{\mathcal{V}_i}$, and this contradicts the assumption that $\mathcal{T}$ is an *extreme* transaction set.

We now consider Case (2), in which there exist some other pairs of transactions τ_c and τ_d $(a \leq c < d \leq n)$ that satisfies $\lceil D_d/T_c \rceil = D_d/T_c$. For this case, we have two subcases to consider: (i) $c = a$, i.e., $\lceil D_d/T_a \rceil = D_d/T_a$ $(b < d)$; (ii) $c > a$, i.e., $\lceil D_d/T_c \rceil = D_d/T_c$. Note that in subcase (ii), d can be smaller than, equal to, or larger than b. For subcase (i), when constructing $\overline{\mathcal{T}}$ by increasing the period of τ_a in Case (1), due to the similar reason as that of processing τ_b, we can get that $\overline{D}_d = D_d$. Hence, the new transaction set $\overline{\mathcal{T}}$ can still be constructed, but $\lceil D_d/T_a \rceil = D_d/T_a$ no longer holds. We thus come to a contradiction to the assumption that $\mathcal{T}$ is an extreme transaction set. For subcase (ii), when constructing $\overline{\mathcal{T}}$ in Case (1), it is clear to see that the deadline and period of τ_d and τ_c would not be impacted. After obtaining $\overline{\mathcal{T}}$, we can repeat a similar process as in Case (1) (by adding a sufficient small positive number to the period of $\overline{\tau}_c$) to construct a new transaction set, say $\overline{\overline{\mathcal{T}}}$, with lower density factor than that of $\overline{\mathcal{T}}$. Consequently, we can also reach a contradiction to the assumption that $\mathcal{T}$ is an *extreme* transaction set.

Based on the above discussions, the lemma follows. □

By Definition 5 and its property presented above, we now introduce an important lemma.

Lemma 3. *Given a transaction set $\mathcal{T} = \{\tau_i\}_{i=1}^n$, if it is $\mathcal{ML}_{DM}$-schedulable with $D_n = T_n$, then $\lambda_\tau \geq \frac{1}{2}$.*

Proof: We consider the case that $\mathcal{T}$ is an *extreme* transaction set. Obviously, if we can prove $\lambda_{\mathcal{T}} \geq \frac{1}{2}$, then by the definition of *extreme* transaction set, we can conclude that all the other transaction sets consisting of n transactions that are $\mathcal{ML}_{DM}$ schedulable with $D_n = T_n$ also have a density factor no less than $\frac{1}{2}$.

From the computation process of $\mathcal{ML}_{DM}$, we know for each transaction τ_i in $\mathcal{T}$, there is $D_i = \sum_{j=1}^{i-1} \lceil D_i/T_j \rceil C_j + C_i$. For presentation convenience, we use k_i^j to denote $\lceil D_i/T_j \rceil$, i.e., $D_i = \sum_{j=1}^{i-1} k_i^j C_j + C_i$. Now based on $\mathcal{T}$, we construct a new transaction set $\mathcal{T}'$. In order to facilitate the distinction, we use C'_i and $\mathcal{V}'_i$ to represent the execution time and validity interval length of transaction τ'_i in $\mathcal{T}'$, and use D'_i and T'_i to denote the corresponding deadline and period of τ'_i. For the worst case execution time of τ'_i $(1 \leq i \leq n-1)$, we consider a modification of the WCET of τ_i $(1 \leq i \leq n-1)$, from C_i to C'_i so that,

$$C'_i = C_i + \varepsilon_i = C_i + \frac{\varepsilon C_i}{\sum_{j=1}^{n-1} C_j/\mathcal{V}_j} \tag{11}$$

where ε is an infinitely small positive value, and let $C'_n = C_n - \varepsilon_n = C_n - \mathcal{V}_n\varepsilon$. It can then be verified that,

$$\sum_{i=1}^{n} \frac{C_i}{\mathcal{V}_i} = \sum_{i=1}^{n} \frac{C'_i}{\mathcal{V}_i} \tag{12}$$

For the validity interval length, we let $\mathcal{V}'_i = \mathcal{V}_i$ $(1 \leq i \leq n-1)$, and let $\mathcal{V}'_n$ be equal to $D'_n + T'_n$, where $T'_n = D'_n$, and D'_n needs to be computed later.

Now with C'_i and $\mathcal{V}'_i$ $(1 \leq i \leq n-1)$, we use $\mathcal{ML}_{DM}$ to derive new deadline and period for each transaction τ'_i $(1 \leq i \leq n-1)$. For τ'_1, it is straightforward to see that $D'_1 = D_1 + \varepsilon_1$ and $T'_1 = \mathcal{V}'_1 - D'_1 = T_1 - \varepsilon_1$, as shown in Table 3. We now proceed to transaction τ'_2. From the computation process of $\mathcal{ML}_{DM}$, we know $D_2 = \lceil D_2/T_1 \rceil C_1 + C_2 = k_2^1 C_1 + C_2$. Since $\mathcal{T}$ is an *extreme* transaction set, by Lemma 2, we know for any two transactions τ_k and τ_j $(k > j)$ in $\mathcal{T}$, there is $\left\lceil \frac{D_k}{T_j} \right\rceil > \frac{D_k}{T_j}$. Hence, $\left\lceil \frac{D_2}{T_1} \right\rceil > \frac{D_2}{T_1}$ obviously holds. Given $\left\lceil \frac{D_2}{T_1} \right\rceil > \frac{D_2}{T_1}$, and ε (and thus ε_1 and ε_2) is sufficiently small, we can get that $\left\lceil \frac{D_2}{T_1} \right\rceil = \left\lceil \frac{D_2 + k_2^1\varepsilon_1 + \varepsilon_2}{T_1 - \varepsilon_1} \right\rceil = k_2^1$. Then, by finding the minimum solution of $\lceil D'_2/T'_1 \rceil C'_1 + C'_2 = D'_2$, we can obtain $D'_2 = D_2 + k_2^1\varepsilon_1 + \varepsilon_2$. Consequently, we have $T'_2 = \mathcal{V}'_2 - D'_2 = \mathcal{V}_i - D'_2 = T_2 - k_2^1\varepsilon_1 - \varepsilon_2$. Following a similar way and due to the same reason, for each transaction τ'_i $(3 \leq i \leq n-1)$, we can compute a new pair

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

of deadline and period to be $D'_i = D_i + \sum_{j=1}^{i-1} k_i^j \varepsilon_j + \varepsilon_i$ and $T'_i = T_i - \sum_{j=1}^{i-1} k_i^j \varepsilon_j - \varepsilon_i$, as shown in Table 3.

Till now, we have finished the process of the first $(n-1)$ transactions of $\mathcal{T}'$, which are $\mathcal{ML}_{DM}$ schedulable with the new computed deadlines and periods. Now with $C'_n = C_n - \mathcal{V}_n\varepsilon$, we use $\mathcal{ML}_{DM}$ to calculate D'_n. We first suppose that D'_n calculated by $\mathcal{ML}_{DM}$ is larger than D_n. By setting $T'_n = D'_n$ and $\mathcal{V}'_n = T'_n + D'_n = 2D'_n$, the following result

$$\begin{aligned}\lambda_{\mathcal{T}'} &= \sum_{i=1}^{n} \frac{C'_i}{\mathcal{V}'_i} = \sum_{i=1}^{n-1} \frac{C'_i}{\mathcal{V}'_i} + \frac{C'_n}{2D'_n} \quad (\text{by } D'_n > D_n) \\ &< \sum_{i=1}^{n-1} \frac{C'_i}{\mathcal{V}_i} + \frac{C'_n}{2D_n} = \sum_{i=1}^{n} \frac{C'_i}{\mathcal{V}_i} \quad (\text{by } \{\mathcal{V}'_i = \mathcal{V}_i\}_{i=1}^{n-1})\end{aligned} \tag{13}$$

can be obtained. Combining Formula (13) with Equation (12), we can get

$$\lambda_{\mathcal{T}'} < \sum_{i=1}^{n} \frac{C'_i}{\mathcal{V}_i} = \sum_{i=1}^{n} \frac{C_i}{\mathcal{V}_i} = \lambda_{\mathcal{T}},$$

which contradicts the assumption that $\mathcal{T}$ is an *extreme* transaction set. Hence, we can conclude that D'_n calculated by $\mathcal{ML}_{DM}$ must be no larger than D_n, i.e., $D'_n \leq D_n$.

Given $\left\lceil \frac{D_n}{T_i} \right\rceil > \frac{D_n}{T_i}$ $(1 \leq i \leq n-1)$ and ε (and thus ε_i $(1 \leq i \leq n)$) is sufficiently small, we have,

$$\left\lceil \frac{D_n}{T_i} \right\rceil = \left\lceil \frac{D_n}{T'_i} \right\rceil = \left\lceil \frac{D_n + \sum_{j=1}^{n-1} k_n^j \varepsilon_j - \varepsilon_n}{T'_i} \right\rceil \quad (1 \leq i \leq n-1) \tag{14}$$

Then, by finding the minimum solution of

$$\sum\nolimits_{j=1}^{n-1} \left\lceil D'_n / T'_j \right\rceil C'_j + C'_n = D'_n \tag{15}$$

we can get $D'_n = D_n + \sum_{j=1}^{n-1} k_n^j \varepsilon_j - \varepsilon_n$. Since $D'_n \leq D_n$, we have $D_n + \sum_{j=1}^{n-1} k_n^j \varepsilon_j - \varepsilon_n \leq D_n$, which means,

$$\begin{aligned}&\varepsilon_n \geq \sum\nolimits_{j=1}^{n-1} k_n^j \varepsilon_j \quad \Rightarrow \quad \mathcal{V}_n\varepsilon \geq \frac{\varepsilon \sum_{i=1}^{n-1} \lceil D_n/T_i \rceil C_i}{\sum_{j=1}^{n-1} C_j/\mathcal{V}_j} \\ &\Rightarrow \mathcal{V}_n \sum\nolimits_{j=1}^{n-1} \frac{C_j}{\mathcal{V}_j} \geq \frac{\mathcal{V}_n}{2} - C_n \;\Rightarrow\; \lambda_{\mathcal{T}} = \sum_{j=1}^{n} \frac{C_j}{\mathcal{V}_j} \geq \frac{1}{2}\end{aligned}$$

The claim is thus proved. □

Based on Lemma 3, we introduce Theorem 4 below, which identifies a sufficient condition for temporal consistency maintenance under DM.

Theorem 4. *Given a transaction set $\mathcal{T}$, if the density factor of $\mathcal{T}$ is not larger than $\frac{1}{2}$, i.e., $\lambda_{\mathcal{T}} \leq \frac{1}{2}$, then $\mathcal{T}$ is temporal consistency schedulable under DM on a uniprocessor system.*

Proof: We prove the claim by contradiction. Suppose $\mathcal{T}$ is NOT temporal consistency schedulable under DM on a uniprocessor system. Then, we know that when using $\mathcal{ML}_{DM}$ to compute deadline and period for some transaction τ_k $(1 \leq k \leq n)$, there is $D_k > \frac{\mathcal{V}_k}{2}$, i.e., $\mathcal{V}_k < 2D_k$. If $k = 1$, then from the computation process of $\mathcal{ML}_{DM}$, we know $D_k = C_k$ and it is clear to see $\lambda_{\mathcal{T}} \geq \frac{C_1}{\mathcal{V}_1} > \frac{1}{2}$; If $k \geq 2$, then by applying Lemma 3 to the first k transactions, we can get

$$\sum_{j=1}^{k} \frac{C_j}{\mathcal{V}_j} = \sum_{j=1}^{k-1} \frac{C_j}{\mathcal{V}_j} + \frac{C_k}{\mathcal{V}_k} > \sum_{j=1}^{k-1} \frac{C_j}{\mathcal{V}_j} + \frac{C_k}{2D_k} \geq \frac{1}{2} \tag{16}$$

which further indicates,

$$\lambda_{\mathcal{T}} = \sum_{i=1}^{n} \frac{C_i}{\mathcal{V}_i} \geq \sum_{j=1}^{k} \frac{C_j}{\mathcal{V}_j} > \frac{1}{2} \tag{17}$$

In summary, we have $\lambda_{\mathcal{T}} > \frac{1}{2}$, which contradicts the assumption that $\lambda_{\mathcal{T}} \leq \frac{1}{2}$, the theorem thus follows. □

To our best knowledge, Theorem 4 is the first result concerning the sufficient condition for temporal consistency maintenance under DM scheduling. It can be seen the sufficient condition for temporal consistency maintenance under DM happens to be the same as that under EDF. Hence, we can design our $\mathsf{TCP_{DM}}$ algorithm, which is the same as $\mathsf{TCP_{EDF}}$, as follows: For any processor M_k, let $\mathcal{T}(M_k)$ denote the transactions from among $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ that have already been allocated to processor M_k. Considering the processors $M_1, M_2, \ldots, M_m$ in any order, $\mathsf{TCP_{DM}}$ assigns transaction τ_i to the first processor M_k, that satisfies the following condition:

$$\lambda_i + \sum\nolimits_{\tau_j \in \mathcal{T}(M_k)} \lambda_j \leq \frac{1}{2} \tag{18}$$

If no such M_k exists, then $\mathsf{TCP_{DM}}$ declares failure: it is unable to conclude that $\mathcal{T}$ is feasible upon the m-processor platform.

Since $\mathsf{TCP_{DM}}$ is essentially the same as $\mathsf{TCP_{EDF}}$, it also has a time complexity of $\mathcal{O}(nm)$, which makes it very time-efficient.

3.2.2 Theoretical Evaluation of $\mathsf{TCP_{DM}}$

We now derive a resource augmentation bound for $\mathsf{TCP_{DM}}$, which characterize its performance. Similar to $\mathsf{TCP_{EDF}}$, we have the following lemma and theorem, which respectively specify a necessary condition and a sufficient condition, for $\mathsf{TCP_{DM}}$ to successfully partition a transaction set. Their proofs are the same as that of Lemma 1 and Theorem 2, respectively, and thus are omitted here for simplicity.

Lemma 4. *If $\mathcal{T}$ is temporal consistency schedulable under DM on an identical multiprocessor platform comprised of m processors each of computing capacity ξ, then we have*

$$\lambda_{max} \leq \frac{1}{2} \cdot \xi \quad and \quad \lambda_{sum} < m \cdot \xi$$

Theorem 5. *Any transaction set $\mathcal{T}$ is successfully scheduled by $\mathsf{TCP_{DM}}$ on m unit-capacity processors, provided that*

$$m \geq \frac{2\lambda_{sum} - 2\lambda_{max}}{1 - 2\lambda_{max}} \tag{19}$$

Based on Lemma 4 and Theorem 5, we have the following resource augmentation result (similar to Theorem 3) regarding Algorithm $\mathsf{TCP_{DM}}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

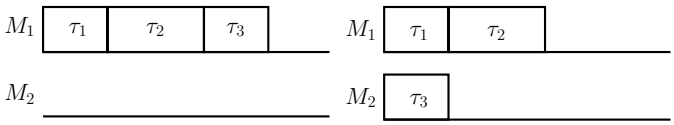

Fig. 2: Transaction Assignment Options 1 and 2.

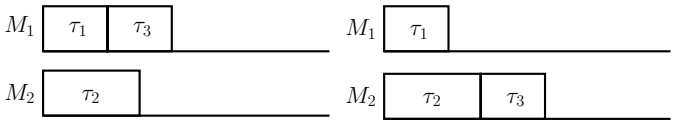

Fig. 3: Transaction Assignment Options 3 and 4.

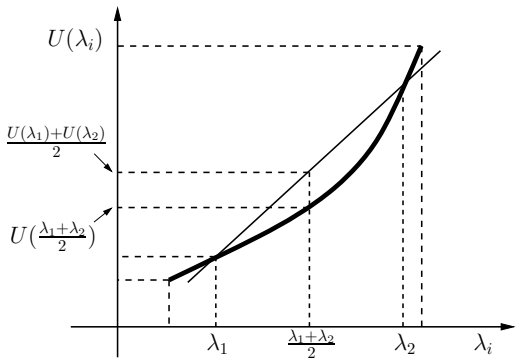

Fig. 4: Balancing density factor leads to lower workload.

Theorem 6. *Algorithm* TCP_{DM} *makes the following performance guarantee: if a transaction set is temporal consistency schedulable on m identical processors each of computing capacity ξ, then* TCP_{DM} *will successfully partition this transaction set upon a platform comprised of m processors that are each $\left(3-\frac{1}{m}\right)$ times as fast as the original.*

4 DBF: A Polynomial-Time Heuristic for W-PARTITION

In this section, we address the W-PARTITION problem globally. As discussed previously in Section 2.5, W-PARTITION is NP-Hard in the strong sense. Hence, the focus of this work is to design workload-efficient transaction assignment schemes which allocate transactions to processors to guarantee temporal consistency in a feasible manner while achieving a total processor workload as low as possible. The period and deadline calculation for transactions on each single processor can be conducted by using $\mathcal{GE}_{EDF}$ and $\mathcal{ML}_{DM}$ under EDF and DM scheduling policies, respectively.

Given the intractability of the W-PARTITION problem, we must look for heuristics. An intuitive way is to modify the four traditional heuristics for the feasibility problem from multiprocessor real-time scheduling, viz next fit (NF), first fit (FF), best fit (BF), and worst fit (WF), to make them applicable to our problem. To facilitate distinction, we use Temporal Consistency Fit, abbreviated TCNF (TCFF, TCBF and TCWF, resp.), to denote the corresponding algorithms which are adopted to solve our problem. The process of TCNF (TCFF, TCBF and TCWF, resp.) is quite similar to their correspondences. The difference comes from that: 1) Inequality (3), rather than the traditional real-time schedulability test, are utilized to conduct the temporal consistency schedulability check when assigning a transaction to a processor; 2) The remaining capacity in TCBF and TCWF is the density factor rather than the utilization in our problem.

But simply applying traditional approaches to our problem may lead to workload-inefficient partitions, as will be illustrated later. To address the W-PARTITION problem in a more workload-efficient way, we first give a simple example to show what dimensions can be got.

Example 1. *Consider three transactions with execution times and validity interval lengths*

$$\mathcal{T} \equiv \{\tau_1 = (2, 16),\ \tau_2 = (3, 17),\ \tau_3 = (2, 30)\}$$

to be executed on $m = 2$ identical processors. It is not difficult to see that any assignment of these transactions to two processors can lead to a feasible schedule under EDF ($\lambda_{sum} \leq 0.5$). If we ignore symmetrical allocations, we have only four possible partitionings (note here after allocating transactions to processors, we use $\mathcal{GE}_EDF$ to deadline and period for them):

1. *All three transactions are allocated to one processor (Figure 2-left): Resulted workload $= 2/14+3/12+2/23 = 0.4798$.*
2. *τ_1 and τ_2 are allocated to one processor and τ_3 is allocated to the other processor (Figure 2-right): Resulted workload $= 2/14 + 3/12 + 2/28 = 0.464$.*
3. *τ_1 and τ_3 are allocated to one processor and τ_2 is allocated to the other processor (Figure 3-left): Resulted workload $= 2/14 + 3/14 + 2/26 = 0.434$.*
4. *τ_2 and τ_3 are allocated to one processor and τ_1 is allocated to the other processor (Figure 3-right): Resulted workload $= 2/14 + 3/14 + 2/25 = 0.437$.*

This simple example with two processors illustrates that workload characteristics of *feasible* partitions can differ significantly: the most workload efficient transaction assignment (partitioning 3) results in about 4% less workload than the first partition. In addition, we observe that the best choice in this example turns out to be the one which yields the most density factor *balanced* partitioning on two processors.

The above example reveals us some useful information. That is, a more density factor balanced partition tends to produce a lower processor workload. Moreover, Figure 4, which is derived from [38], illustrates why a density factor balanced partitioning is more possible to achieve a lower workload.

From Figure 4, we can see that the workload generated on a single processor versus the total density is like a convex curve. Given two transaction sets with density factors λ_1 and λ_2 ($\lambda_1 < \lambda_2$) on two different processors, let the resulting processor workload be $U(\lambda_1)$ and $U(\lambda_2)$, respectively. Moreover, let the processor workload corresponds to the transaction set with density

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

TABLE 4: Partition Results for Example 2

TCNF / TCFF / TCBF								TCWF								DBF							
M_1				M_2				M_1				M_2				M_1				M_2			
τ_i	C_i	D_i	T_i	τ_i	C_i	D_i	T_i	τ_i	C_i	D_i	T_i	τ_i	C_i	D_i	T_i	τ_i	C_i	D_i	T_i	τ_i	C_i	D_i	T_i
τ_1	2	2	7	τ_3	2	2	14	τ_1	2	2	7	τ_3	3	3	8	τ_1	2	2	7	τ_2	3	3	8
τ_2	3	5	6	τ_4	1	3	15	τ_2	2	4	12	τ_4	1	4	14	τ_3	2	4	12	τ_5	3	6	18
				τ_5	3	6	18	τ_6	2	6	34	τ_5	3	7	17	τ_4	1	5	13				
				τ_6	2	8	32									τ_6	2	7	33				
$\lambda = 0.495$				$\lambda = 0.3556$				$\lambda = 0.39722$				$\lambda = 0.45328$				$\lambda = 0.45277$				$\lambda = 0.3977$			
Total workload: 1.2244								*Total workload*: 1.1341								*Total workload*: 1.13157							

factor $\frac{\lambda_1+\lambda_2}{2}$ on a single processor be $U\left(\frac{\lambda_1+\lambda_2}{2}\right)$. If the convexity holds, it can be observed from Figure 4 that $2 \times U\left(\frac{\lambda_1+\lambda_2}{2}\right) < U(\lambda_1) + U(\lambda_2)$, which means a density factor balanced partitioning on two processors leads to a lower workload. Consequently, in order to decrease the total processor workload as much as possible, it is better to balance density factor among all processors to the greatest extent. Note that the above convexity assumption is based on our observations instead of formal proofs.

Based on the above discussion, we propose our heuristic Density factor Balancing Fit as follows: Considering the processors $\{M_1, M_2, \ldots, M_m\}$ in any order, Algorithm DBF assigns transaction τ_i to the first processor M_k that satisfies condition (3) and the following condition:

$$\lambda_i + \sum\nolimits_{\tau_j \in \mathcal{T}(M_k)} \lambda_j \leq \frac{\lambda_{sum}}{m} \tag{20}$$

If no such M_k exists, then DBF assigns transaction τ_i to the first processor M_k which satisfies condition (3). If again no such M_k exists, DBF declares failure: it is unable to conclude that the transaction set $\mathcal{T}$ is feasible on the m-processor platform. Detail of DBF is shown in Algorithm 1.

The proposed heuristic DBF is efficient from the two performance dimensions: feasibility and workload. We will detail its performance in Section 5. But at first, it should be noted that DBF has the same resource augmentation bound as $\mathsf{TCP}_{\mathsf{EDF}}$ and $\mathsf{TCP}_{\mathsf{DM}}$, which guarantees its feasibility performance theoretically.

Theorem 7. *Algorithm DBF preserves a resource augmentation bound of* $(3 - \frac{1}{m})$.

Proof: Since only condition (3) is used to check feasibility, while condition (20) is for balancing density factor, the claim follows directly. □

Complexity: In attempting to allocate transaction τ_i, observe that DBF essentially evaluates, in (20) and (3), the density factor of the previously allocated $(i-1)$ transactions on each of the m processors. Since these values can be computed in constant time, the run-time of Algorithm DBF is no more than $\mathcal{O}(nm)$.

The following example illustrates the advantage of DBF compared to the four schemes evolved from traditional methods for multiprocessor scheduling.

Algorithm 1: DBF: Density factor Balancing Fit

Input : A set of update transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{n}$.
Output: Assigning each τ_i to a processor.

```
for i = 1; i ≤ n; i = i + 1 do
    for j = 1; j ≤ m; j = j + 1 do
        if τ_i satisfies Conditions (20) and (3) on M_j then
            assign τ_i to M_j;
            break and proceed to transaction τ_{i+1};
    if τ_i is not assigned to any processor then
        for k = 1; k ≤ m; k = k + 1 do
            if τ_i satisfies Condition (3) on M_k then
                assign τ_i to M_k;
                break and proceed to transaction τ_{i+1};
    if τ_i is not assigned to any processor then
        Return PARTITIONING FAILED;
```

Example 2. *Consider a transaction set comprised of six transactions with execution times and validity interval lengths*

$$\begin{aligned}\mathcal{T} \equiv\ & \{\tau_1 = (2, 9), \tau_2 = (3, 11), \tau_3 = (2, 16),\\ & \tau_4 = (1, 18), \tau_5 = (3, 24), \tau_6 = (2, 40)\}\end{aligned}$$

to be executed on $m = 2$ identical processors. The resulted solutions under TCNF, TCFF, TCBF, TCWF and DBF are stated in Table 4, with all periods and deadlines derived by $\mathcal{GE}_{EDF}$. Note here for the given transaction set, TCNF, TCFF and TCBF produce the same result.

Example 2 demonstrates that density factor balancing plays an important role in minimizing the processor workload. As can be seen from Table 4, the most density factor balanced partition, i.e., the one-derived by DBF, results in about 9% less workload than TCNF, TCFF and TCBF, the least density factor balanced partitions. Note here TCWF produces almost the same workload as DBF. In fact, TCWF has a better workload performance than DBF in most cases. This is because TCWF tends to distribute the density factor evenly among all the processors, and thus can produce density factor balanced partitions. But it should also be noted that TCWF has a higher run-time complexity than DBF, and its feasibility performance is bad, as will be shown in the experiment study.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

TABLE 5: Experimental parameters and settings

Para. Class	Para(s)	Meaning	Value
System	N_{CPU}	No. of CPU	{2,4,8,16,32}
	N_T	No. of data objects	[10,500]
	$\mathcal{V}_i$(ms)	Validity interval of x_i	[4000,8000]
Update Transactions	C_i(ms)	Time for updating x_i	[5-15,15-150,150-800]
	Length	No. of data to update	1

5 Performance Evaluation

In this section, we provide an experimental evaluation of DBF versus TCNF, TCFF, TCBF and TCWF. Notice here $\mathcal{GE}_{EDF}$ is used to compute deadline and period of update transactions for all algorithms evaluated in the experiments. Moreover, in addition to the theoretical evaluation of DBF's feasibility aspect stated in Theorem 7, we also evaluate DBF by comparing it with an algorithm FF-GE, a variant of First-Fit which uses $\mathcal{GE}_{EDF}$ as the feasibility check when allocating a transaction to a processor. The aim of this evaluation is to characterize the schedulability loss of DBF due to using a sufficient - but not necessary - polynomial time feasibility test on each processor.

5.1 Simulation Model and Assumptions

Performance Metrics: Our problem has two equally important performance dimensions: temporal consistency schedulability (or feasibility) and processor workload. Given a transaction set to be scheduled on a multiprocessor platform, an algorithm with high temporal consistency feasibility performance, low workload, and low computational cost is favorable. But as we will see later shortly, there is an inherent trade-off between feasibility and workload performances of the schemes we investigated. Hence, judging by the feasibility and workload it is not always possible to point to a "clear" winner. Consequently, we define an additional hybrid metric (namely feasibility/workload) that combines both feasibility and workload performance dimensions. In summary, we measure the performance of a given heuristic $\mathcal{H}$ in terms of the following three metrics:

1) The *Feasibility Performance* ($FP_{\mathcal{H}}$), given as the percentage of the transaction sets that are schedulable by $\mathcal{H}$.
2) The *Workload Performance* ($WP_{\mathcal{H}}$), given as average workload of transaction sets that are scheduled by $\mathcal{H}$ in feasible manner.
3) The *Feasibility/Workload* ($FW_{\mathcal{H}}$) metric, given as $\frac{FP_{\mathcal{H}}}{WP_{\mathcal{H}}}$.

Notice that metric 3) favors the heuristics with high feasibility performance and low processor workload.

Simulation Settings: Table 5 shows a summary of the parameters and default settings used in our experiments. Note here we use similar baseline values for the parameters as [37] and [38], which are originally from air traffic control applications [28], to keep consistency and continuity with previous work. Two categories of parameters are defined: system and update transaction. For system configurations, an m (selected from $\{2, 4, 8, 16, 32\}$) processors, main memory based RTDBS is considered. The number of real-time data objects N_T ranges from 10 to 500 to generate different density factor loads in the system. The validity interval length $\mathcal{V}_i$ of each real-time data object is assumed to be uniformly distributed in $[4000, 8000]$. For update transactions, it is assumed that each update transaction updates one data object, and each transaction has a uniform probability of having short (5-15ms), medium (15-150ms), or long (150-800ms) execution time. Observe that the range of execution time implicitly means the maximum density factor for individual transaction is $\lambda_{max} = 800/4000 = 0.2$.

We have generated a total of 100000 transaction sets by varying the number of processors m, the total density factor λ_{sum} of the update transaction set, and the number of transactions n. We considered systems with 2, 4, 8, 16 and 32 processors while generating transaction sets with different number of transactions which range in $[10, 500]$. Note that for simplicity, we present our results only in the context of 50-200 transactions that are to be scheduled on 8 processors, however we must underline that the trends and relative performances of techniques are similar in other settings as well.

All the algorithms to be evaluated are implemented in C++. For each point plotted in the figure, the simulations continued until a confidence interval of 95% with half-width of less than 5% about the mean was achieved.

5.2 Experimental Results

Figure 5 shows the feasibility performance. It can be seen that under low to medium (about 2.5) density factor, feasibility can be easily achieved and all heuristics yield 100% feasibility. As density factor increases, the feasibility performance of all schemes drops sharply, and eventually it becomes zero (when λ_{sum} exceeds 4). This is because all the five heuristics use condition (3) as the feasibility test, which is restricted to the number of processors. It can be seen that TCWF has the worst feasibility performance, while DBF, TCFF and TCBF have almost the same performance on feasibility. Figure 6 presents the workload performance. As can be observed, TCWF obtains the best workload performance, followed by DBF, of which is a bit higher compared to TCWF. Also note that TCNF, TCFF and TCBF have almost the same and, worst workload performance. The largest gap between TCWF and TCFF is about 60%. This is because these algorithms greedily schedule the transactions on one processor to the extent it is possible while keeping other processors idle, and this results in unbalanced density factor partitionings in many cases. It is also interesting to note that TCFF and TCBF are hardly distinguishable in both workload and feasibility dimensions in this set of experiments. By examining the performance of the five heuristics, we observe that DBF is by far the best heuristic in terms of overall performance: Its feasibility performance is the best (Figure 5); Moreover, its workload performance, though is higher than TCWF, clearly dominates TCNF, TCFF and TCBF, throughout

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

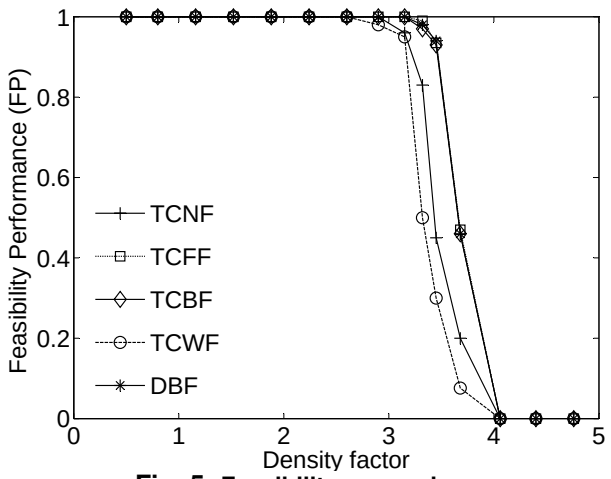

Fig. 5: Feasibility comparison

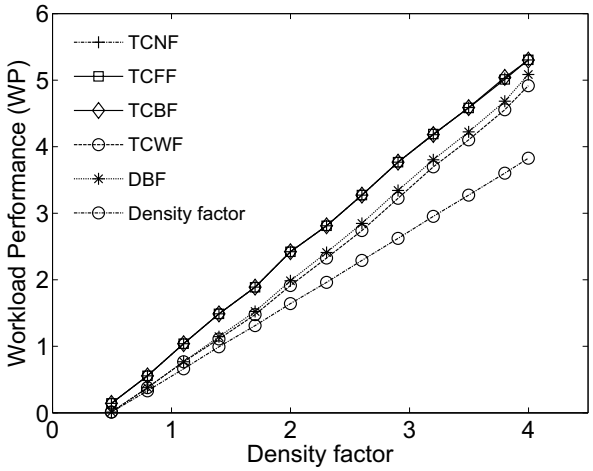

Fig. 6: Workload comparison

the entire density factor spectrum (Figure 6). This fact is even more emphasized by the feasibility/workload curves of heuristics (Figure 7).

In fact, much of the performance differences among the partitioning heuristics, in terms of both feasibility and workload, can be explained in terms of their density factor balancing behavior. TCFF and TCBF tend to yield *unbalanced* partitions, while TCWF and DBF tend to produce *balanced* ones. This different density factor-balancing behavior leads to different feasibility and workload characteristics. On one hand, by distributing the density factor evenly among the available processors can result in balanced partitions and thus lead to a relatively lower workload. On the other hand, by greedily packing as many transactions as possible on a few processors (just in the case of TCFF and TCBF), it is possible to accommodate additional transactions on the remaining (idle) processors and thus improve the feasibility. So there is intrinsically a trade-off between feasibility and workload performance, and it would be better to design a heuristic which can balance these two factors in a certain degree. DBF happens to be such a choice. It combines both the advantage of TCWF and TCFF. The experimental result also verifies DBF's better performance compared to other heuristics.

Figure 8 shows the feasibility performance comparison between DBF and FF-GE. Under low to medium density factor parameters, feasibility can be easily achieved and both heuristics yield 100% feasibility. As density factor load increases, the feasibility performance of DBF drops sharply, and eventually it becomes zero (when λ_{sum} exceeds 4). But FF-GE can still derive feasible solutions until λ_{sum} approaches to 5.5. It can be seen that FF-GE can schedule a larger number of transaction systems than DBF for all distributions we have tested. This is because according to Theorem 1, any transaction set with $\lambda_{sum} \leq 0.5$ can be scheduled by $\mathcal{ML}_{DM}$. Hence, for a transaction set, if DBF can produce a feasible solution, then FF-GE can also derive one. But the converse is not true. This also illustrates why FF-GE has better feasibility performance than DBF.

In summary, DBF shows its advantage on workload performance and low computation cost (polynomial-time complexity). But its feasibility is somewhat restricted to the sufficient feasibility test, i.e., $\lambda_{sum} \leq 0.5m$. Nevertheless, DBF still guarantees a resource augmentation bound of $3 - \frac{1}{m}$.

6 Related Work

There has been a lot of work on RTDBSs for maintaining real-time data freshness [10], [12], [17]–[19], [21]–[23], [32]–[34]. [34] studies the performance of two well known concurrency control algorithms, two-phase locking and optimistic, in maintaining temporal consistency of shared data in a hard real-time systems. [22] investigates real-time data-semantics and proposes a class of real-time access protocol called SSP (Similarity Stack Protocol). The trade-off between data consistency and system workload is exploited in [16], where similarity-based principles are combined with the *Half-Half* scheme to reduce workload by skipping the execution of task instances. [12] focuses on maintaining data freshness in soft real-time embedded systems and proposes an algorithm (ODTB) for updating data items that can skip unnecessary updates allowing for better CPU utilization.

All the work mentioned above assumes the deadlines and periods of update transactions are given, hence gives no answer to the period and deadline assignment problem for maintaining temporal consistency. To address the period and deadline assignment problem, the *More-Less* scheme is proposed in [37] with *Deadline Monotonic* scheduling. While *More-Less* is based on periodic task model, the deferrable scheduling algorithm for fixed priority transactions (*DS-FP*) proposed in [36] follows a sporadic task model. *DS-FP* reduces processor workload by adaptively adjusting the separation of two consecutive instances of update transactions while satisfying the validity constraint. [17] investigates how to maintain the mutual temporal consistency of real-time data objects. [14] studies the problem of how to maintain the temporal validity of real-time data in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

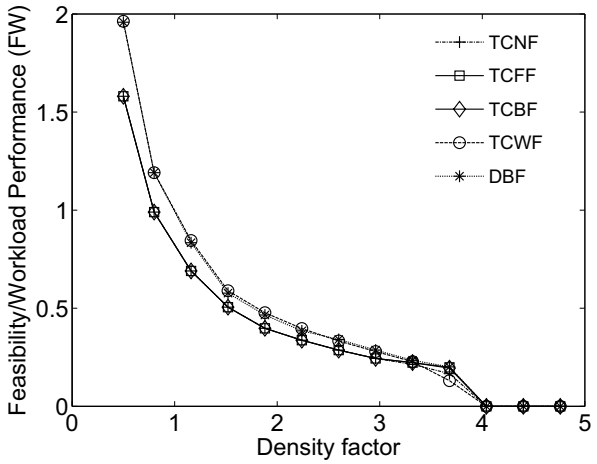

Fig. 7: Feasiblity/Workload comparison

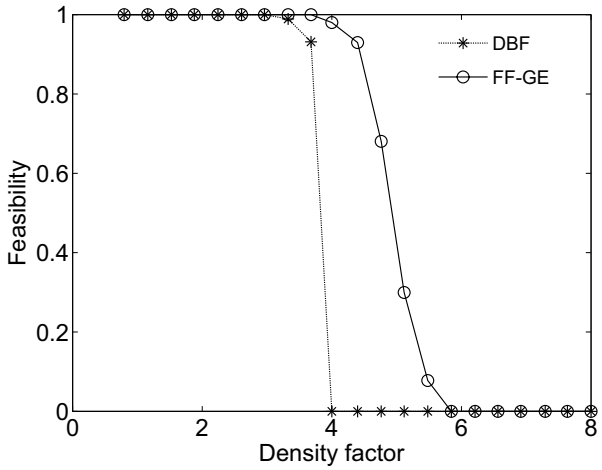

Fig. 8: Feasibility loss comparison

presence of mode changes in flexible real-time systems. The authors propose to use different scheduling policies in different modes and introduce two algorithms to search for proper switch points. The period and deadline assignment problem for real-time update transactions scheduled under EDF is firstly addressed in [38], and later an improved solution is given in [26]. Recently, Han et al. [15] and Wang et al. [35] studied the co-scheduling problem of periodic update and application transactions and proposed several effective scheduling algorithms.

Most of the related work mentioned above focuses on uniprocessor systems. The first work considers real-time temporal issue on multiprocessor platforms is the one by Lundberg [29], which focuses on age-constraint global multiprocessor scheduling. Our work differs from [29] in that we address partitioned scheduling.

7 Conclusions

In this paper, we studied the workload-aware transaction partitioning problem for maintaining temporal consistency of real-time data objects upon multiprocessor platforms. As far as we know, this work is the first attempt to solve the given problem. We first only considered the feasibility aspect of the problem by proposing two polynomial-time partitioning algorithms $\mathsf{TCP}_{\mathsf{EDF}}$ and $\mathsf{TCP}_{\mathsf{DM}}$ under EDF and DM scheduling, respectively. We formally proved that the resource augmentation bound of both $\mathsf{TCP}_{\mathsf{EDF}}$ and $\mathsf{TCP}_{\mathsf{DM}}$ are $3 - \frac{1}{m}$. Secondly, we addressed the problem holistically and developed our density factor balancing scheme DBF, showing that a more balanced partitioning tends to produce a lower workload. Our experimental evaluation demonstrates that DBF is by far the best choice from the feasibility/workload performance point of view.

For future work, we intend to investigate the temporal consistency scheduling problem on multiprocessors with aperiodic task model, and the resource sharing issue. We also plan to study the temporal consistency scheduling problem upon global multiprocessor platforms.

Acknowledgments

A preliminary version of this paper appeared in [25], the 32nd IEEE Real-Time Systems Symposium, pages 126-135, Vienna, Austria, 2011. The authors would like to thank the anonymous reviewers for their constructive and helpful comments. This work was substantially supported by the State Key Program of National Natural Science of China under Grant No. 61332001, National Natural Science Foundation of China under Grants Nos. 61173049, 61300045, and China Postdoctoral Science Foundation under Grant No. 2013M531696.

References

[1] B., S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proc. of IEEE Real-Time Systems Symp.*, pages 193–202, 2001.

[2] B., K. Bletsas, and S. Baruah. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *Proc. of IEEE Real-Time Systems Symp.*, pages 385–394, 2008.

[3] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proc. of IEEE Real-Time Systems Symp.*, pages 120–129, 2003.

[4] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proc. of IEEE Real-Time Systems Symp.*, pages 119–128, 2007.

[5] S. Baruah and N. Fisher. The Partitioned Multiprocessor Scheduling of Sporadic Task Systems. In *Proc. of IEEE Real-Time Systems Symp.*, pages 321–329, 2005.

[6] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Trans. on Parallel and Distributed Systems*, 20(4):553–566, 2009.

[7] J.-J. Chen and S. Chakraborty. Resource augmentation bounds for approximate demand bound functions. In *Proc. of IEEE Real-Time Systems Symp.*, pages 272–281, 2011.

[8] R. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.

[9] N. Fisher, S. Baruah, and T. Baker. The Partitioned Scheduling of Sporadic Tasks According to Static-Priorities. In *Proc. of Euromicro Conference on Real-Time Systems*, pages 118–127, 2006.

[10] R. Gerber, S. Hong, and M. Saksena. Guaranteeing end-to-end timing constraints by calibrating intermediate processes. In *Proc. of IEEE Real-Time Systems Symp.*, pages 192–203, 1994.

[11] J. Goossens, S. Baruah, and S. Funk. Real-time scheduling on multiprocessor. In *Proc. of theInternational Conference on Real-Time System*, pages 189–204, 2002.

[12] T. Gustafsson and J. Hansson. Data management in real-time systems: a case of on-demand updates in vehicle control systems. In *Proc. of IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 182–191, 2004.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2448088, IEEE Transactions on Computers

[13] S. Han, D. Chen, M. Xiong, and A. Mok. A Schedulability Analysis of Deferrable Scheduling Using Patterns. In *Proc. of Euromicro Conference on Real-Time Systems*, pages 47–56, 2008.

[14] S. Han, D. Chen, M. Xiong, and A. Mok. Online Scheduling Switch for Maintaining Data Freshness in Flexible Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symp.*, pages 115–124, 2009.

[15] S. Han, K.-Y. Lam, J. Wang, K. Ramamritham, and A. K. Mok. On co-scheduling of update and control transactions in real-time sensing and control systems: Algorithms, analysis, and performance. *IEEE Trans. on Knowledge and Data Engineering*, 25(10):2325–2342, 2013.

[16] S. Ho, T. Kuo, and A. Mok. Similarity-based load adjustment for real-time data-intensive applications. In *Proc. of IEEE Real-Time Systems Symp.*, pages 144–154, 1997.

[17] A. Jha, M. Xiong, and K. Ramamritham. Mutual Consistency in Real-Time Databases. In *Proc. of IEEE Real-Time Systems Symp.*, pages 335–343, 2006.

[18] K.-D. Kang, S. H. Son, and J. A. Stankovic. Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE Trans. on Knowledge and Data Engineering*, 16(10):1200–1216, 2004.

[19] K.-D. Kang, Y. Zhou, and J. Oh. Estimating and Enhancing Real-Time Data Service Delays: Control Theoretic Approaches. *IEEE Trans. on Knowledge and Data Engineering*, 57(2), 2009.

[20] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In *Proc. of IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 23–32, 2009.

[21] Y. K. Kim and S. H. Son. Predictability and consistency in real-time database systems. *Advances in real-time systems*, pages 509–531, 1993.

[22] T. Kuo and A. Mok. SSP: A semantics-based protocol for real-time data access. In *Proc. of IEEE Real-Time Systems Symp.*, pages 76–86, 1993.

[23] K. Lam, M. Xiong, B. Liang, and Y. Guo. Statistical Quality of Service Guarantee for Temporal Consistency of Real-Time Data Objects. In *Proc. of IEEE Real-Time Systems Symp.*, pages 276–285, 2004.

[24] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[25] J. Li, J.-J. Chen, M. Xiong, and G. Li. Workload-aware partitioning for maintaining temporal consistency upon multiprocessor platforms. In *Proc. of IEEE Real-Time Systems Symp.*, pages 126–135, 2011.

[26] J. Li, M. Xiong, V. C. Lee, L. Shu, and G. Li. Workload-efficient deadline and period assignment for maintaining temporal consistency under edf. *IEEE Trans. on Computers*, 62(6):1255–1268, 2013.

[27] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[28] D. Locke. Real-Time Databases: Real-World Requirements. *Kluwer International Series In Engineering and Computer Science*, pages 83–92, 1997.

[29] L. Lundberg. Multiprocessor Scheduling of Age Constraint Processes. In *Proc. of IEEE Embedded and Real-Time Computing Systems and Applications*, pages 42–47, 1998.

[30] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Cambridge, MA, USA, 1983.

[31] K. Ramamritham. Real-time databases. *Distributed and Parallel Databases*, 1(2):199–226, 1993.

[32] K. Ramamritham. Where Do Time Constraints Come From? Where Do They Go? *Journal of Database Management*, 7:4–11, 1996.

[33] K. Ramamritham, S. H. Son, and L. C. Dipippo. Real-time databases and data services. *Real-Time Systems*, 28(2):179–215, 2004.

[34] X. Song and J. Liu. Maintaining temporal consistency: pessimistic vs. optimistic concurrency control. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):786–796, 1995.

[35] J.-T. Wang, K.-Y. Lam, S. Han, S. H. Son, and A. K. Mok. An effective fixed priority co-scheduling algorithm for periodic update and application transactions. *Computing*, 95(10-11):993–1018, 2013.

[36] M. Xiong, S. Han, K. Lam, and D. Chen. Deferrable scheduling for maintaining real-time data freshness: algorithms, analysis, and results. *IEEE Trans. on Computers*, pages 952–964, 2008.

[37] M. Xiong and K. Ramamritham. Deriving Deadlines and Periods for Real-Time Update Transactions. *IEEE Trans. on Computers*, pages 567–583, 2004.

[38] M. Xiong, Q. Wang, and K. Ramamritham. On earliest deadline first scheduling for temporal consistency maintenance. *Real-Time Systems*, 40(2):208–237, 2008.

Jianjun Li received the PhD degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China in 2012. He was a Senior Research Associate with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. He is currently a post-doctoral and lecturer at the School of Computer Science and Technology in HUST. His research interests include real-time systems, real-time databases and advanced data management.

Jian-Jia Chen is a Professor in the Department of Informatics in TU Dortmund University in Germany. He was a Juniorprofessor in the Department of Informatics in Karlsruhe Institute of Technology (KIT) in Germany from May 2010 to March 2014. He received his Ph.D. degree from Department of Computer Science and Information Engineering, National Taiwan University, Taiwan in 2006. He received his B.S. degree from the Department of Chemistry at National Taiwan University 2001. Between Jan. 2008 and April 2010, he was a postdoc researcher at Computer Engineering and Networks Laboratory in ETH Zurich, Switzerland. His research interests include real-time and embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing. He received Best Paper Awards from ACM SAC in 2009, IEEE RTCSA in 2005 and 2013, and IEEE/ACM CODES+ISSS in 2014.

Ming Xiong received the B.S. degree in computer science and engineering from Xian Jiaotong University, M.S. degree in computer science from Sichuan University, China, and the Ph.D. degree in computer science from University of Massachusetts, Amherst. Dr. Xiong's research interests include real-time systems, database systems, and mobile computing. From 2000 to 2009, he was a member of the Technical Staff at Bell Laboratories Research, Lucent Technologies, in Murray Hill, New Jersey. He is currently a member of the Technical Staff at Google, Inc.

Guohui Li received the PhD degree in computer science from Huazhong University of Science and Technology (HUST), China in 1999. He was promoted to a full professor in 2004, and currently acts as vice dean of the School of Computer Science and Technology in HUST. His research interests mainly include real-time systems, mobile computing, and advanced data management.

Wei Wei received the Ph.D. degree from Huazhong University of Science and Technology (HUST), China, in 2012. He is currently an assistant professor with the School of Computer of Science and Technology in HUST. He was a Research Fellow with Nanyang Technological University, and Singapore Management University, Singapore. His research interests include information retrieval, data mining, and real-time computing.