Workload-Aware Partitioning for Maintaining Temporal Consistency upon Multiprocessor Platforms

Jianjun Li[†], Jian-Jia Chen[‡], Ming Xiong[§] and Guohui Li[†]✉

[†]*School of Computer Science and Technology,*
Huazhong University of Science and Technology (HUST), Wuhan, China, 430074
[‡]*Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 76131* [§]*Google Inc., USA*
jianjunli@smail.hust.edu.cn, j.chen@kit.edu, mxiong@google.com, guohuili@hust.edu.cn (✉)

Abstract

Deriving deadlines and periods of update transactions for maintaining timeliness and data freshness has long been recognized as an important problem in real-time database research. Despite years of active research, the state of the art only focuses on uniprocessor systems. In this paper, we take a first step of studying 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 earliest deadline first (EDF) scheduling, while minimizing the total workload on m processors. Firstly, we only consider the feasibility aspect of the problem by proposing a polynomial time partitioning scheme, Temporal Consistency Partitioning (TCP), and formally showing that the resource augmentation bound of TCP is $(3 - \frac{1}{m})$. 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 Databases; Temporal Consistency; Deadline and Period; Partitioning; Multiprocessor*

I. Introduction

Real-Time database systems (RTDBS) have been widely used in many applications that require timely processing of large amounts of real-time data, 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 transactions. An object in the database models the current status of a real world entity in the external environment, for example, the longitude and latitude 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 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. Sensor transactions are generated by intelligent sensors which periodically sample the values of real-time objects. When sensor 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 a few important reasons to minimize the processor workload imposed by sensor update transactions [16], [29], [30]: (1) it helps save sensor energy; (2) the RTDBS is able to process more sensor update transactions; and (3) more processor capacity can be left to other application transactions that are triggered due to environmental changes committed by update transactions.

In the past, while there has been much work devoted to the temporal consistency scheduling problem, almost all of them focus on uniprocessor systems. Some examples are Half-Half (*HH*) [16], More-Less [30], *DS-FP* [14], [29], and $\mathcal{HS}_{EDF}$ [31]. In this work, we take a first step to address the workload-aware temporal consistency maintenance problem on multiprocessor platforms. Our aim is to design efficient partition schemes which can allocate 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 update transactions on the platform.

Multiprocessor scheduling of periodic tasks is one of the most extensively studied area in real-time systems research. In general, the approaches fall into either *global* or *partitioned*

1052-8725/11 $26.00 © 2011 IEEE
DOI 10.1109/RTSS.2011.19

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], [19] 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].

From workload-awareness perspective, the immediate advantage of concentrating on partitioned multiprocessor scheduling is the ability to apply well-established uniprocessor temporal consistency scheduling schemes (like $\mathcal{ML}_{EDF}$, $\mathcal{ML}_{DM}$, etc.) once transaction-to-processor assignments are determined. Another obvious benefit of adopting partitioned approach stems from the fact that well-established *deferrable scheduling schemes* for aperiodic systems can be readily adopted on each processor to further decrease workload once the transaction allocation is made. In contrast, *deferrable scheduling* for global aperiodic multiprocessor scheduling is an open problem to a considerable extent. We believe that avoiding the overhead of transaction migration is another incentive to focus on the partitioned approach.

This work: We consider the problem of workload minimization for periodic preemptive real-time update transactions that are scheduled on an identical multiprocessor platform. We adopt partitioned scheduling and assume that transactions are assigned dynamic (earliest deadline first) priorities in each single processor. Partitioned multiprocessor real-time scheduling considers *feasibility* as the main objective. The problem is 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. Our work is based on the observation that the transaction allocation can have a significant impact on the overall workload of the system. In particular, we find that a density factor[1] balanced partition scheme tends to result in lower workload. The main contribution of this paper can be summarized as follows:

1) We first consider the feasibility aspect of the workload-aware transaction assignment problem by proposing a polynomial-time partition algorithm, Temporal Consistency Partition (TCP), and formally show that the resource augmentation bound of TCP is $(3 - \frac{1}{m})$.
2) We address the workload-efficient transaction-to-processor assignment problem globally by characterizing it as a density factor balance problem and proposing a polynomial time heuristic, Density factor Balancing Fit (DBF).
3) 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 II, 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 II gives the definition of temporal validity and presents some notations and assumptions. The problem to be addressed is also introduced. In Section III, we present and evaluate the performance of a polynomial-time partitioning scheme TCP for the partition problem when only considering the feasibility aspect. Section IV addresses the problem globally by detailing the design of our heuristic DBF. Evaluation results of DBF versus other heuristics are described in Section V. Section VI briefly reviews some related work and finally, conclusions are drawn in Section VII.

II. 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 $\mathcal{HS}_{EDF}$, which will be used as the period and deadline selection scheme on each single processor. Finally, we define the problem to be addressed in this work.

A. Temporal Validity for Data Freshness

In a real-time database, a data object is a logic 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 [23], [26], [27].

Definition 1. *[26] A real-time data object (x_i) at time t is temporally valid if, for (any of) 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, the corresponding update transaction 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

TABLE I: 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 $\{\tau_i\}_{i=1}^n$
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 $\{\tau_i\}_{i=1}^n$
$\mathcal{M}$	A set of multiprocessors $\{M_1, M_2, \ldots M_m\}$

problem of how to assign periods and deadlines to update transactions to maintain the validity consistency while minimizing the utilization on a single processor. Among these approaches, *More-Less* [30] 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 $\mathcal{V}_i = T_i + D_i$.

B. 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 partitioned multiprocessor scheduling. Transactions are assigned *permanently* to processors. On each processor, we use $\mathcal{HS}_{EDF}$ [31], a search-based More-Less scheme, which is by far the best choice under EDF, 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 a distinct data object, 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. As D_i and T_i are to be determined by our schemes, we choose to consider *constrained* transaction deadlines, 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 $\mathcal{T}$ is $U_{sum} = \sum_{i=1}^n \frac{C_i}{T_i}$ and $\lambda_{sum} = \sum_{i=1}^n \frac{C_i}{\mathcal{V}_i}$, respectively. For presentation simplicity, we use $\mathcal{T}(M_k)$ to denote the transactions that have been assigned to M_k. Lastly, preemptive scheduling algorithms are assumed. The symbols used in this paper are presented in Table I.

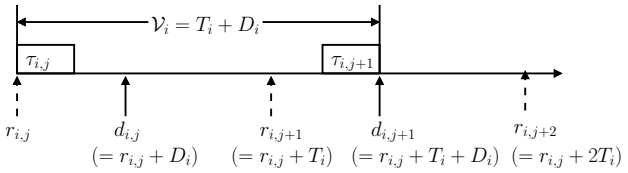

Fig. 1: Illustration of *More-Less* scheme.

C. $\mathcal{HS}_{EDF}$: A Heuristic More-Less Scheme On a Single Processor

To derive periods and deadlines which can guarantee temporal validity of data objects while minimizing the update workload on a single processor, $\mathcal{HS}_{EDF}$ is proposed by Xiong et al. [31], in which EDF is used to schedule periodic update transactions. $\mathcal{HS}_{EDF}$ is a search-based heuristic and is capable of finding a solution if one exists. $\mathcal{HS}_{EDF}$ starts by setting all transactions' period to $\mathcal{V}_i - C_i$ $(1 \leq i \leq n)$, and then checks the schedulability of the initial solution. If this solution is feasible, the algorithm terminates and returns it as the final solution. Otherwise, $\mathcal{HS}_{EDF}$ involves solving a selection problem to determine which transaction's period should be decreased. The selection problem is a *knapsack problem* which is solved by using a *branch and bound* method. After that, a schedulability test is conducted on the new solution to check its feasibility. The above process is repeated until a solution is found to be feasible. Full description of $\mathcal{HS}_{EDF}$ is omitted here due to space limitation, where the details can be found in [31].

D. Problem Definition

Our aim in this paper 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 deadlines and periods for transactions on each single processor by a period deadline selection scheme, such that:

1. *the transactions assigned to each processor can be scheduled in a feasible manner, and*
2. *the total workload on $\mathcal{M}$ 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 [25]. The reduction is as follows: For each task, we generate a corresponding real-time 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 the W-PARTITION is NP-Complete in the strong sense. With the optimization of total workload, W-PARTITION is NP-Hard in the strong sense.

III. TCP: Temporal Consistency Partition

Considering the intractability of the problem, we first do not take the workload issue into consideration, but only focus on how to derive feasible transaction partitioning. Specifically, we present a polynomial-time partition algorithm: *Temporal Consistency Partition* (TCP) in Section III-A, and provide the theoretical evaluation in Section III-B.

A. Design of TCP: Temporal Consistency Partition

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 *if, for each transaction, a couple of period and deadline can be derived by a period and deadline selection algorithm (e.g. $\mathcal{HS}_{EDF}$) 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 on uniprocessor systems, and thus paves the way for our design of TCP.

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 deemed to be temporal consistency schedulable on a uniprocessor system.*

Proof: Given that $\lambda_{sum} \leq 0.5$, it is obvious for each transaction τ_i, we can derive a couple 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,

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

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

Based on Theorem 1, we design the temporal consistency partition 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 TCP assigns transaction τ_i to a processor M_k, that satisfies the following condition:

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

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

Since Theorem 1 provides a sufficient condition for any transaction set to be temporal consistency schedulable on a single processor, it is straightforward to conclude that TCP can guarantee the assignment is feasible if it succeeds to return a partition on transaction set $\mathcal{T}$. TCP is quite time-efficient due to that when allocating one transaction τ_i, it only need to evaluate the density factor of the previously allocated $(i-1)$ transactions on each of the m processors by (2). Since this value can be computed in constant time, it is obvious that the run-time of TCP in assigning all n transactions is no more than $\mathcal{O}(nm)$. Next, we offer a quantitative evaluation of the efficacy of Algorithm TCP.

B. Theoretical Evaluation of TCP

Resource augmentation (or speedup factor) 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 evaluation method, 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 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 its validity interval length.

Below, we derive a resource augmentation bound (upper bound here) for Algorithm TCP, which characterize its performance. We first present a useful lemma.

Lemma 1. *If $\mathcal{T}$ is temporal consistency schedulable on an identical multiprocessor platform comprised of m processors each of computing capacity ξ, then it must be the case that*

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

Proof: For each transaction τ_i, there is $C_i \leq D_i \cdot \xi \leq T_i \cdot \xi$. Given that $\mathcal{V}_i = T_i + D_i$, we have,

$$\lambda_i = \frac{C_i}{\mathcal{V}_i} = \frac{C_i}{T_i + D_i} \leq \frac{C_i}{2D_i} \leq \frac{1}{2} \cdot \xi. \tag{3}$$

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, then its density factor must 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 Algorithm TCP (or in fact, any partition algorithm) to successfully partition a transaction set. We now present a theorem below, which specifies a sufficient condition for TCP to successfully partitioning a transaction set.

Theorem 2. *Any transaction set $\mathcal{T}$ can be successfully scheduled by TCP on m unit-capacity processors, given that*

$$m \geq \frac{2\lambda_{sum} - 2\lambda_{max}}{1 - 2\lambda_{max}}. \quad (4)$$

Proof: We prove this by considering the case when TCP fails to assign τ_i. Then, it must be the case that on each processor M_k $(1 \leq k \leq m)$, there is

$$\lambda_i + \sum\nolimits_{\tau_j \in \mathcal{T}(M_k)} \lambda_j > \frac{1}{2}. \quad (5)$$

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

$$m\lambda_i + \sum\nolimits_{\tau_j \in \mathcal{T}(\mathcal{M})} \lambda_j > \frac{1}{2} \cdot m. \quad (6)$$

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

$$(m-1)\lambda_i + \lambda_{sum} > \frac{1}{2} \cdot m \;\Rightarrow\; m < \frac{2\lambda_{sum} - 2\lambda_i}{1 - 2\lambda_i}. \quad (7)$$

Therefore, when

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

TCP can successfully schedule $\mathcal{T}$ on m processors. ■

By Theorem 2, we now present a resource augmentation result regarding TCP.

Theorem 3. *TCP can guarantee the following performance: if a transaction set is temporal consistency schedulable on m identical processors each of computing capacity ξ, then TCP will successfully 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 on m processors each of computing capacity ξ. We will prove below that $\mathcal{T}$ is guaranteed to be successfully partitioned by TCP on m unit-capacity processors for $\xi \leq \frac{m}{3m-1}$.

Since $\mathcal{T}$ is temporal consistency schedulable 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 (4), 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. ■

It should be noted that the bound derived above is not tight. The main reason comes from that $\lambda_{sum} < m \cdot \xi$ is pessimistic for necessary condition of feasibility.

IV. DBF: A Polynomial-Time Heuristic for W-PARTITION

In this section, we address the W-PARTITION problem globally. As discussed in Section II, W-PARTITION is NP-Hard in the strong sense. Hence, in this work, we concentrate on how 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 computation for transactions on each single processor is conducted by using the $\mathcal{HS}_{EDF}$ algorithm, which is by far the best choice [31] under EDF for reducing the workload on uniprocessor systems.

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 (2), rather than the traditional real-time schedulability test, is 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*

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

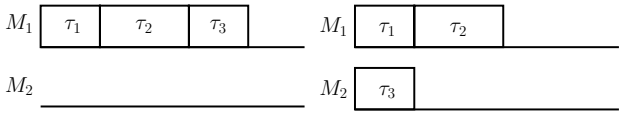

Fig. 2: Transaction Assignment Options 1 and 2.

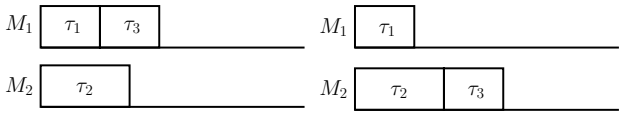

Fig. 3: Transaction Assignment Options 3 and 4.

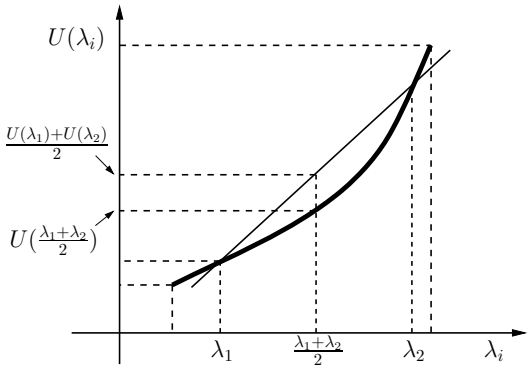

Fig. 4: Balancing density factor leads to lower workload.

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:

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 [31], illustrates why a density factor balanced partitioning is more possible to achieve a lower workload.

Based on Figure 4, we can observe that the workload generated on a single processor versus the total density is like a convex curve. If the convexity holds, given two density factors λ_1 and λ_2 on two different processors, let the resulting processor workload be $U(\lambda_1)$ and $U(\lambda_2)$, respectively, while the processor workload corresponds to density factor $\frac{\lambda_1+\lambda_2}{2}$ be $U\left(\frac{\lambda_1+\lambda_2}{2}\right)$, it can be seen from Figure 4 that $U\left(\frac{\lambda_1+\lambda_2}{2}\right) < \frac{U(\lambda_1)+U(\lambda_2)}{2}$. 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.

Algorithm 1: DBF: Density factor Balancing Fit

Input : A set of update transactions $\mathcal{T} = \{\tau_i\}_{i=1}^{n}$ sorted in non-decreasing order of $\mathcal{V}_i$.
Output: Assigning each transaction 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 (8) and (2) on M_j then
            assign τ_i to M_j;
            break and proceed to next 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 (2) on M_k then
                assign τ_i to M_k;
                break and proceed to next transaction τ_{i+1};
    if τ_i is not assigned to any processor then
        /* Fail to derive a solution. */
        Return PARTITIONING FAILED;
```

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_j that satisfies inequality (2) and the following condition:

$$\lambda_i + \sum\nolimits_{\tau_j \in \mathcal{T}(M_k)} \lambda_j \leq \frac{\lambda_{sum}}{m} \tag{8}$$

If no such M_j exists, then DBF assigns transaction τ_i to the first processor M_k which satisfies condition (2). 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

TABLE II: 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							

performance in Section V. But at first, it should be noted that DBF has the same resource augmentation bound as TCP, which identifies a feasibility performance guarantee theoretically.

Theorem 4. *Algorithm DBF preserves a resource augmentation bound of* $(3 - \frac{1}{m})$.

Proof: Since only inequality (2) is used to check feasibility, while inequality (8) is for balancing density factor, the claim follows directly. ■

Complexity of DBF: When allocating transaction τ_i, observe that DBF essentially evaluates, in (8) and (2), 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, it is clear that the run-time of the algorithm in allocating all n transactions 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.

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

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

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

This example demonstrates that density factor balancing plays an important role in minimizing the processor workload. As can be seen from Table II, 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.

TABLE III: Experimental parameters and settings

Para. Class	Parameters	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]
	Trans.length	No. of data to update	1

V. Performance Evaluation

In this section, we provide an experimental evaluation of DBF versus TCNF, TCFF, TCBF and TCWF. In addition to the theoretical evaluation of DBF's feasibility aspect stated in Theorem 4, we also evaluate DBF by comparing it with an algorithm FF-HS, a variant of First-Fit which uses $\mathcal{HS}_{EDF}$ 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.

A. 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, there is an inherent tradeoff 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:

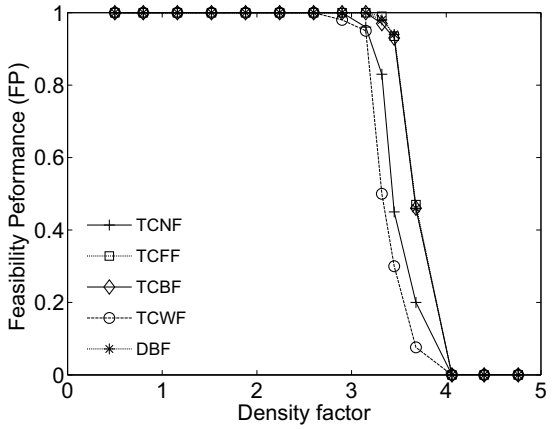

Fig. 5: Feasibility performance

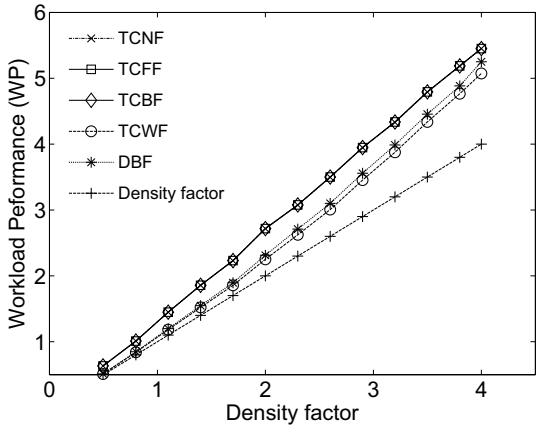

Fig. 6: Workload performance

(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/Worklaod* ($FW_{\mathcal{H}}$) metric, given as $\frac{FP_{\mathcal{H}}}{WP_{\mathcal{H}}}$.

It is not difficult to see that $FW_{\mathcal{H}}$ favors the heuristics with high feasibility performance and low processor workload.

Simulation Settings: Table III shows a summary of the parameters and default settings used in our experiments. Note here we use similar baseline values for the parameters as [30] and [31], which are originally from air traffic control applications [23], 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, a 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 $[4000, 8000]$ms. 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 transaction 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 $[10, 500]$. Due to the lack of space, 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.

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.

B. 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). Among the five heuristics, TCWF has the worst feasibility performance, while DBF, TCFF and TCBF have almost the same, and the best, feasibility performance. 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. The remaining three, i.e., TCNF, TCFF and TCBF have almost the same and, worst workload performance. The largest gap between the most workload-efficient (TCWF) and the least workload-efficient (TCFF, TCWF, TCNF) is about 60%. 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 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

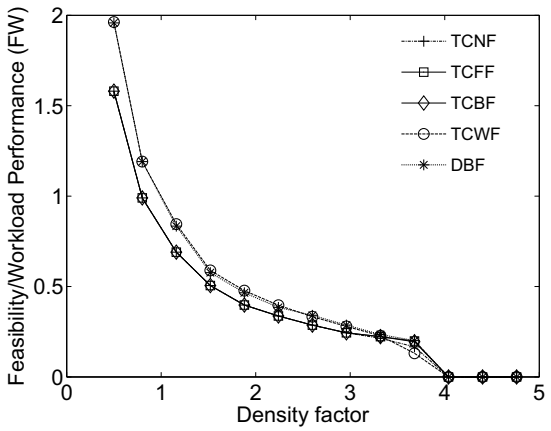

Fig. 7: Feasiblity/Worklaod performance

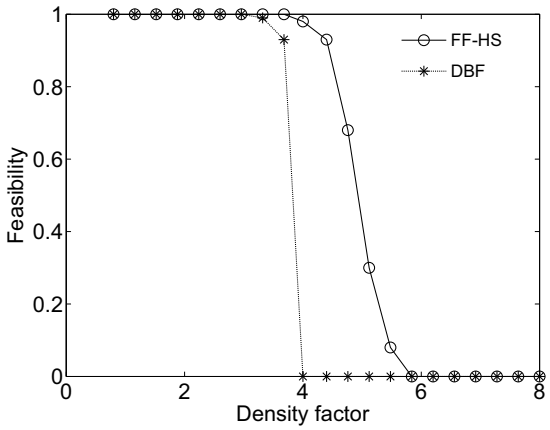

Fig. 8: Feasibility loss comparison

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. In view of the intrinsical tradeoff between feasibility and workload performance, 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. As can be observed, DBF combines both the advantage of TCWF and TCFF. First, when allocating one transaction, it checks whether the density factor exceeds $\frac{\lambda_{sum}}{m}$ to guarantee density factor balancing, and hence can result in a relatively lower workload. This policy is similar to that of TCWF, which always allocates one transaction to the processor with maximum remaining capacity to balance density factors among all processors. Second, DBF places as many transactions as possible on one processor when a balance cannot be achieved, with the objective of accommodating more transactions on a single processor to improve feasibility. This behavior is analog to that of TCFF, which always selects the first processor that verifies the schedulability test. In summary, the experimental result verifies DBF's better feasibility/workload performance compared to the other four heuristics.

Figure 8 shows the feasibility performance comparison between DBF and FF-HS. 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-HS can still derive feasible solutions until λ_{sum} approaches to 5.8. This is because according to Theorem 1, any transaction set with $\lambda_{sum} \leq 0.5m$ is deemed to be temporal consistency schedulable, and hence can also be scheduled by FF-HS. But the converse is not true, which means when λ_{sum} exceeds $0.5m$, FF-HS may still find a solution, while DBF definitely fails. This illustrates why FF-HS has better feasibility performance than DBF.

VI. Related Work

There has been a lot of work on RTDBSs for maintaining real-time data freshness [10], [12], [17], [18], [20]–[22], [27], [28]. In [28], Song et al. study 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. In [21], Kuo et al. investigate real-time data-semantics and propose 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. In [13], Gustafsson et al. focus on maintaining data freshness in soft real-time embedded systems and propose an on-demand scheduling algorithm (ODDFT) for guaranteeing the freshness of base and derived data. In [12], Gustafsson et al. propose 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 [30] 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 [29] 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.

In [17], Jha et al. investigate how to maintain the mutual temporal consistency of real-time data objects. In [15], Han et al. study the problem of how to maintain the temporal validity of real-time data in the 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 addressed in [31].

Most of the previous work mentioned above focuses on uniprocessor systems. To our best knowledge, the only work considers temporal issue on multiprocessor platforms is the one conducted by Lundberg [24], which focuses on age-constraint global multiprocessor scheduling. Our work is different from [24] in that we address partitioned scheduling.

VII. 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 and proposed a polynomial-time partitioning algorithm TCP. We formally proved that the resource augmentation bound of TCP is $(3-\frac{1}{m})$. Secondly, we addressed the problem globally by proposing a 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 perspectives of feasibility and workload.

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

Acknowledgments

We are grateful to the anonymous reviewers for their constructive comments. This research was partially supported by the National Science Foundation of China [Award No. 60873030] and the Research Fund for the Doctoral Program of the Ministry of Education of China [Award No. 20090142110023].

References

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

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

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

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

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

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

[7] J.-J. Chen and S. Chakraborty. Resource Augmentation Bounds for Approximated Demand Bound Functions. In *Proc. of IEEE RTSS*, 2011.

[8] R. Davis and A. Burns. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. *University of York, Department of Computer Science, Tech. Rep. YCS-2009-443*, 2009.

[9] N. Fisher, S. Baruah, and T. Baker. The Partitioned Scheduling of Sporadic Tasks According to Static-Priorities. In *Proc. of 18th 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 *Proceedings of IEEE Real-Time Systems Symposium*, pages 192–203, 1994.

[11] J. Goossens, S. Baruah, and S. Funk. Real-time scheduling on multiprocessor. In *Proc. of International Conference on Real-Time System*, 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 RTAS*, pages 182–191, 2004.

[13] T. Gustafsson and J. Hansson. Dynamic on-demand updating of data in real-time database systems. In *Proc. of ACM symposium on Applied Computing*, pages 846–853. New York, USA, 2004.

[14] S. Han, D. Chen, M. Xiong, and A. Mok. A Schedulability Analysis of Deferrable Scheduling Using Patterns. In *Proc. of ECRTS*, pages 47–56, 2008.

[15] 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 Symposium*, pages 115–124, 2009.

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

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

[18] K. Kang, S. Son, J. Stankovic, and T. Abdelzaher. A QoS-Sensitive Approach for Timeliness and Freshness Guarantees in Real-Time Databases. In *Proc. of Euromicro Conference on Real-Time Systems*, 2002.

[19] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In *Proc. of RTAS*, pages 23–32, 2009.

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

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

[22] 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 Symposium*, 2004.

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

[24] L. Lundberg. Multiprocessor Scheduling of Age Constraint Processes. In *Proceedings of RTCSA*, page 42, 1998.

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

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

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

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

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

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

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