The Journal of Systems and Software 85 (2012) 1729–1743

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Adaptive co-scheduling for periodic application and update transactions in real-time database systems

Song Han [a], Kam-yiu Lam [b,*], Jiantao Wang [b], Sang H. Son [c], Aloysius K. Mok [a]

[a] Department of Computer Science, University of Texas at Austin, United States
[b] Department of Computer Science, CityU of Hong Kong, Hong Kong
[c] Department of Computer Science, University of Virginia, United States

ARTICLE INFO

Article history:
Received 31 October 2011
Received in revised form 15 February 2012
Accepted 24 March 2012
Available online 10 April 2012

Keywords:
Real-time database systems
Data freshness
Real-time co-scheduling
Update generation and processing

ABSTRACT

In this paper, we study the co-scheduling problem of periodic application transactions and update transactions in real-time database systems for surveillance of critical events. To perform the surveillance functions effectively, it is important to meet the deadlines of the application transactions and maintain the quality of the real-time data objects for their executions. Unfortunately, these two goals are conflicting and difficult to be achieved at the same time. To address the co-scheduling problem, we propose a real-time co-scheduling algorithm, called *Adaptive Earliest Deadline First Co-Scheduling* (*AEDF-Co*). In *AEDF-Co*, a dynamic scheduling approach is adopted to adaptively schedule the update and application jobs based on their deadlines. The performance goal of *AEDF-Co* is to determine a schedule for given sets of periodic application and update transactions such that the deadline constraints of all the application transactions are satisfied and at the same time the *quality of data* (*QoD*) of the real-time data objects is maximized. Extensive simulation experiments have been performed to evaluate the performance of *AEDF-Co*. The results show that by adaptively adjusting the release times of update jobs and scheduling the update and application jobs dynamically based on their urgencies, *AEDF-Co* is effective in achieving the performance goals and maximizing the overall system performance.

© 2012 Elsevier Inc. All rights reserved.

1. Introduction

Most previous studies on *real-time database systems* are concentrated on soft real-time systems, such as stock trading systems (Adelberg et al., 1995; Amirijoo et al., 2006; Kang et al., 2004) and traffic information systems, with the main performance goal to maximize the *average* performance of the systems, e.g., minimize the number of deadline missing transactions or maximize the freshness of data objects (Shanker et al., 2008; Kang et al., 2009; Ramamritham et al., 2004; Lam and Kuo, 2001; Lam et al., 2002; Abbott and Garcia-Molina, 1992). With the rapid advances in wireless communication and sensor technologies in recent years, real-time database technologies are gaining increasing interests for applications into various time-critical surveillance systems. Example applications include robot control (Dertouzos, 1974), health monitoring systems (Ko et al., 2010), industrial plant management (Li and Liu, 2009; Song et al., 2008) and vehicular control (Gustafsson and Hansson, 2004). With the use of database technologies, various sophisticated database operations and functions can be supported to enhance the surveillance functions. Unlike soft real-time database systems, the main performance goal of these surveillance systems is to provide a guarantee in performance in monitoring the operation environment for generating timely and appropriate responses to react to critical events occurring in the environment. Similar to hard real-time systems, if the responses cannot be generated timely, the consequences could be serious and may even be catastrophic.

To perform effective surveillance functions in an operation environment with the use of a real-time database system, a common method is to use *periodic application transactions* which are defined according to application and surveillance requirements (Nixon et al., 2008; Gustafsson and Hansson, 2004). Each invocation of an application transaction (called an application job in this paper) has a hard or firm deadline on its completion time. How to schedule a set of periodic application jobs to meet their deadlines is a typical real-time scheduling problem (Liu and Layland, 1973). However, simply satisfying the real-time constraints of the application transactions are not sufficient to achieve the surveillance functions effectively. Another issue of equal importance is to maintain the freshness or called *temporal validity* (Ramamritham, 1993) of the set of real-time data objects for their executions.

* Corresponding author.
E-mail addresses: shan@cs.utexas.edu (S. Han), cskylam@cityu.edu.hk (K.-y. Lam), jiantwang2@student.cityu.edu.hk (J. Wang), son@virginia.edu (S.H. Son), mok@cs.utexas.edu (A.K. Mok).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.03.055

Real-time data objects are defined to describe the current status of dynamic entities in an operation environment such as the location of a moving object and the temperature of a production engine (Ramamritham et al., 2004; Shanker et al., 2008). Unlike the data objects in conventional database systems, a real-time data object has a temporal validity constraint (Ramamritham, 1993). It is temporally valid (or simply called *valid* in the rest of the paper) if its latest value is "truly" reflecting the current status of the corresponding entity in the operation environment. Otherwise, it is invalid or called "stale" (Golab et al., 2009; Bateni et al., 2009). As the status of an entity in the operation environment can be highly dynamic and change continuously with time, a real-time data object will become stale with the passage of time (Ramamritham, 1993). Accessing stale data can seriously affect the *quality of service* (*QoS*) provided by the application transactions, i.e., generate incorrect responses even though the responses are timely (Xiong et al., 2010; Golab et al., 2009; Kang et al., 2004).

How to generate and schedule update transactions to maintain the temporal validity and freshness of real-time data objects has been a hot research topic (Xiong and Ramamritham, 2004; Ho et al., 1997; Golab et al., 2009; Labrinidis and Roussopoulos, 2001; Qu and Labrinidis, 2007; Kang et al., 2004; Xiang et al., 2008). One of the common methods is to use *periodic update transactions*. Each invocation of an update transaction is called an update job. New data values are generated periodically by sensors installed in an operation environment. Once generated, it is sent to a scheduler to refresh the validity of the corresponding data object maintained in the database. The main design goal is to maintain the validity of a set of real-time data objects with minimum total update workload.

In this paper, instead of studying the update generation and scheduling problems of update transactions for maintaining data validity (Xiong and Ramamritham, 2004; Ho et al., 1997) or studying efficient real-time scheduling methods for periodic application transactions to meet their deadlines (Liu and Layland, 1973), we perform an integrated study of the two problems. We call the problem as the *real-time co-scheduling* problem in a real-time database system. In particular, we concentrate on the co-scheduling problem in time-critical surveillance systems in which a real-time scheduler is responsible for processing periodic application transactions to monitor a set of dynamic entities through the sensor data values received from them. An example of such applications is shown in Section 4.1. They require meeting the deadlines of all the application transactions and at the same time maintaining the data validity for the execution of application transactions. However, how to achieve these two performance goals at the same time is a big research challenge. As will be discussed in the related work, previous proposed methods such as Kang et al. (2007, 2009) are not sufficient for many time-critical surveillance systems, e.g., industrial plant management where "correct" and timely responses need to be generated for efficient process management and event detection. They aim to provide just a *soft guarantee* in data quality and meeting the deadline constraints of application transactions using various heuristics in scheduling.

Obviously, the scheduling of update and application transactions is conflicting with each other as they compete for the same set of resources for execution. In this paper, we concentrate on the scheduling of the CPU as it is the most important resource in supporting real-time processing (Liu and Layland, 1973). In addition, the database is assumed to reside at the main memory to eliminate any I/O delay. This assumption is valid for most real-time database systems as the number of data objects to be maintained in a real-time database is usually not large (Xiong and Ramamritham, 2004; Ramamritham et al., 2004).

Intuitively, scheduling the update transactions at higher priorities compared with the application transactions can maximize the data validity but the probability of violating the deadline constraints of the application transactions will be higher. On the other hand, scheduling the application transactions first can maximize the probability of meeting their deadline constraints but the *quality of data* (*QoD*) could be seriously degraded. Thus, in this paper, we propose a novel dynamic co-scheduling algorithm, called *Adaptive Earliest Deadline First Co-scheduling* (*AEDF-Co*). The performance goal of *AEDF-Co* is to determine a schedule such that the deadline constraints of all the application transactions are met and at the same time the *QoD* of the real-time data objects is maximized. With the application of *AEDF-Co*, a deterministic solution can be obtained in the design phase of a system. One of the main intuitions behind *AEDF-Co* is that in cases we cannot meet the deadline constraints of all the application transactions, we may extend the validity intervals of some of the data objects to reduce the impacts of update jobs scheduling on the performance of the application jobs. The technical challenge is how to minimize the overall impact to the QoD of the data objects. In this paper, we use a dynamic scheduling to maximize the total QoD in the system as well as the total quality of service (QoS) obtained from the application transactions. The performance of *AEDF-Co* is studied through extensive simulation and the results show that *AEDF-Co* is effective in achieving the performance goals as compared with the baseline methods, such as the Update First (*UF*) and Application First (*AF*) schemes under different update and application transaction workloads.

The remainder of the paper is organized as follows. In Section 2, we briefly review the important previous research on maintaining freshness and temporal validity of real-time data objects and co-scheduling of update and application transactions. In Section 3, we first discuss the data validity and the staleness of real-time data. We then review the basic principles of one of the best known periodic method called *More–Less* (*ML*) for maintaining data validity and its limitations when it is used for co-scheduling. In Section 4, we first give a motivating example to explain the co-scheduling problem and then introduce a new metric to measure the quality of data (*QoD*) for processing of periodic application transactions. The details of the proposed co-scheduling algorithm *AEDF-Co* are presented in Section 5. Section 6 reports the major findings from the performance studies. We conclude the paper and discuss the future work in Section 7.

2. Related works

In recent years there has been extensive research on maintaining temporal validity and freshness of real-time data objects (Labrinidis and Roussopoulos, 2001; Xiong and Ramamritham, 2004; Xiong et al., 2005, 2006a, 2008a, 2010; Han et al., 2008, 2009; Lam et al., 2004; Ahmed and Vrbsky, 2000; Gustafsson and Hansson, 2004; Xiang et al., 2008). Many of the papers use periodic update transactions to provide a guarantee in data validity, i.e., the data objects are valid throughout the whole operation period of a system. One of the earliest proposals is the Half–Half (*HH*) method (Ho et al., 1997) in which the period and relative deadline of an update transaction are each set to be half of the validity interval (Ramamritham, 1993) of the data object to be updated. To further reduce the update workload, the More–Less (*ML*) method (Xiong and Ramamritham, 2004), which will be reviewed in Section 3, was proposed.

In contrast to *HH* and *ML*, the deferrable scheduling algorithm for fixed-priority transactions (*DS-FP*) (Xiong et al., 2008a) follows the aperiodic task model in generating update jobs. It exploits the semantics of temporal validity constraint of real-time data objects by judiciously deferring the release times of update jobs. To reduce the online scheduling overhead of *DS-FP*, two extensions were

proposed to produce a hyperperiod for *DS-FP* so that the schedule generated by repeating the hyperperiod infinitely will satisfy the validity constraints of the real-time data objects (Xiong et al., 2008b). In Xiong et al. (2008b), based on a sufficient feasibility condition of EDF scheduling, it proposed an extension of *ML* called ML_{EDF} which is a linear time algorithm to solve the period and deadline assignment problem.

Instead of providing a complete guarantee in data validity, some works adopted a less restrictive criterion such as statistical guarantee in quality of service (QoS) (Amirijoo et al., 2006; Kang et al., 2004; Xiong et al., 2006b; Wei et al., 2006). In Wei et al. (2006), a prediction-based QoS management scheme for periodic queries over dynamic data streams was proposed. In Xiong et al. (2006b), several extensions of *ML* called statistical *ML* algorithms were proposed to provide a statistical guarantee in data validity for systems where the processing time of an update transaction may vary following a pre-defined distribution.

Various heuristic-based dynamic scheduling methods were proposed for soft real-time database systems where the arrivals of update jobs are sporadic and unpredictable. Some of the proposals are (Golab et al., 2009; Bateni et al., 2009; Labrinidis and Roussopoulos, 2001). Unlike the validity constraint proposed in Ramamritham (1993), in Golab et al. (2009), it was defined that a data value starts to be stale right after its generation. Based on the earliest deadline first (EDF) scheduling algorithm, it proposed a maximum benefit with look-ahead algorithm to assign scheduling priorities to the update jobs in an update stream with the purpose to minimize the total staleness of data objects in a system. In Bateni et al. (2009), in addition to minimizing the total data staleness, it aimed to limit the maximum stretch. By assuming a quasi-periodic model in update generation, a constant-stretch algorithm was proposed. In Labrinidis and Roussopoulos (2001), the scheduling problem of update streams to maintain freshness of dynamic information such as those contained in dynamic web pages was studied. Based on the popularities and dependencies of views, a quality-aware update scheduling algorithm (QoDA) was proposed to maximize the quality of data. In QoDA, an update with the biggest negative impact on the quality of data is given the highest priority for processing. In Ahmed and Vrbsky (2000) and Gustafsson and Hansson (2004), various on-demand schemes were proposed to reduce the number of unnecessary updates to minimize the CPU utilization for processing the updates.

The co-scheduling problem of update and application transactions has been receiving growing interests in recent years. In Qu and Labrinidis (2007), a two-level co-scheduling algorithm called query update time sharing was proposed in which at the higher level it dynamically adjusts the query and update share of CPU to maximize the overall system profit. At the lower level, queries and updates have their own priority queues. The co-scheduling of updates and queries was also studied in Thiele et al. (2009). However, unlike Qu and Labrinidis (2007), it is for systems where the arrivals of updates and queries are pre-defined. Based on the pure update first and query first methods, it proposed an optimal schedule for given sets of updates and queries to maximize both quality of data and quality of service. In Kulkarni et al. (2008), the execution of queries over dynamic update streams was studied and a load-adaptive scheme was proposed based on the rate monotonic scheduling. In Kang et al. (2004, 2007, 2009), feedback control-based methods were proposed for dealing with unpredictable system workload and changing data requirements of real-time transactions. They apply feedback-based miss ratio control schemes to maintain the performance within a desired value. The main problem of these previous studies on solving the co-scheduling problem is that they are not suitable for real-time surveillance systems which require guarantees in both meeting the deadline constraints of application transactions and data quality to ensure the correct execution of the surveillance functions performed by the application transactions.

Table 1
Symbols and definitions.

Symbol	Definition
X_i	Real-time data object i
τ_i^u	Update transaction i for updating X_i
$\mathcal{V}_i^{min}$	Minimum validity interval of X_i and τ_i^u
$\mathcal{V}_i^{max}$	Maximum validity interval of X_i and τ_i^u
$\mathcal{V}_i$	Validity interval of X_i and τ_i^u
$\mathcal{T}^u$	The set of update transactions
$\mathcal{T}^a$	The set of application transactions
τ_i^a	Application transaction i
$C_i^u(C_i^a)$	Worst-case execution time (WCET) of $\tau_i^u(\tau_i^a)$
$D_i^u(D_i^a)$	Relative deadline of $\tau_i^u(\tau_i^a)$
$P_i^u(P_i^a)$	Period of $\tau_i^u(\tau_i^a)$
$J_{i,j}^u(J_{i,j}^a)$	The $j+1th$ job of transaction $\tau_i^u(\tau_i^a)$ $(j=0, 1, 2, \ldots)$
$r_{i,j}^u(r_{i,j}^a)$	Release time of $J_{i,j}^u(J_{i,j}^a)$
$d_{i,j}^u(d_{i,j}^a)$	Absolute deadline of $J_{i,j}^u(J_{i,j}^a)$
$S_i(t)$	Staleness of data object X_i at time t
$\mathcal{Q}_i^u(t)$	Quality of data (QoD) of object X_i at time t
$\mathcal{Q}_i^a(t)$	Quality of service (QoS) of application transaction τ_i^a at time t

3. Preliminaries and motivation

In this section, we first discuss the validity and staleness of real-time data. In Section 3.2, we review the More–Less (*ML*) method which is the basic method for maintaining data validity using periodic update transactions. Table 1 summarizes the symbols that are used frequently in the paper.

3.1. Validity and staleness of real-time data

In Ramamritham (1993), an "interval-based" approach was proposed to define the temporal validity constraint for a real-time data object. It is assumed that the "maximum rate of changes" in status of a dynamic entity ζ_i is bounded and its accuracy requirement is pre-defined from applications. Based on these two parameter values, a validity interval $\mathcal{V}_i$ is defined for data object X_i such that the difference in "value" between the status of an entity ζ_i and the corresponding data object X_i is not larger than the accuracy requirement within the validity interval, $\mathcal{V}_i$.

Definition 3.1. A real-time data object X_i at time t is valid if its validity interval $\mathcal{V}_i$ plus the release time t_r of the update job from which it obtains its current value is not less than t, that is, $t_r + \mathcal{V}_i \geq t$ (Ramamritham, 1993).

The importance of the validity constraint is that if a data object accessed by an application job is valid, the degree of currency of the data object is guaranteed (Golab et al., 2009). Another important property of this criterion is that it can easily be combined with the concept of data similarity (Kuo and Mok, 1993) for resolving the concurrency control problem between update and application transactions. Under the concept of data similarity, the values from two successive update jobs for the same data object will be considered to be "similar" if the difference in their release times is smaller than the validity interval (Kuo and Mok, 1993). If there is a data conflict between an update job and an application job in accessing the data object, the serialization order between the update job and the application job can be reversed to make the schedule to be serializable. Therefore, it is not necessary to have a concurrency control mechanism for resolving the data conflicts between these two types of transactions (Kuo and Mok, 1993, 1994).

Following the above criterion for data validity and the assumption on the "maximum rate of changes" in status of a dynamic entity, we define the *staleness* of a real-time data object.

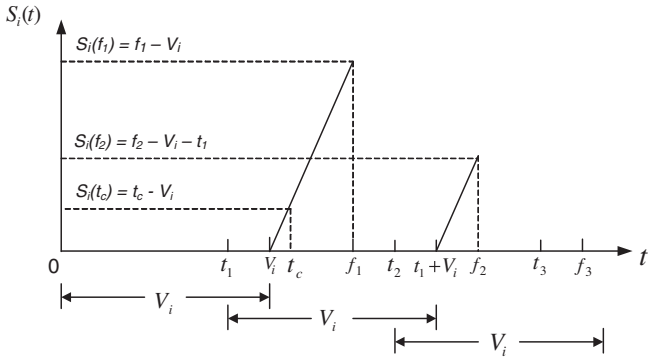

Fig. 1. An illustration of the staleness of data object X_i.

Definition 3.2. Assume that the current value of data object X_i is released at time t_r and its validity interval is $\mathcal{V}_i$. The staleness of X_i at time t is defined as:

$$S_i(t) = \begin{cases} 0, & t \in [t_r, t_r + \mathcal{V}_i] \\ t - (t_r + \mathcal{V}_i), & t > t_r + \mathcal{V}_i \end{cases}$$

Note that under this definition, the minimum value of staleness of a data object is zero while its maximum value is unbounded and increases with time continuously until the completion of the next update job. As shown in Fig. 1, data object X_i with validity interval of length $\mathcal{V}_i$ is initialized at time 0. An update job is generated at time t_1 and the new value is installed at time f_1 due to preemption from higher priority jobs. The staleness of X_i increases linearly in interval $[\mathcal{V}_i, f_1)$ and drops to 0 at time f_1.

In this paper, it is assumed that the quality of a data object is quantified by its staleness. If the staleness of a data object is larger, its quality will be lower. To ensure correct execution of the application transactions, it is one of the main performance goals of this paper to minimize the data staleness observed by the application transactions.

3.2. The More–Less method and its problems

More–Less (*ML*) (Xiong and Ramamritham, 2004) is an *off-line deterministic* method using periodic update transactions to maintain the temporal validity of a set of real-time data objects. To maintain the validity of object X_i, an update transaction τ_i^u generates a job $J_{i,j}^u$ at the beginning of every fixed period P_i^u after capturing the latest status of the entity ζ_i. With the given validity interval of each data object, *ML* determines the periods and relative deadlines for a set of update transactions and schedules their jobs using *Deadline Monotonic* (*DM*) (Liu and Layland, 1973). The calculation order for each update transaction follows the *Shortest Validity First* (*SVF*) principle so that it starts from the update transaction with the shortest validity interval. In calculating the relative deadline and period for an update transaction, it adopts a pessimistic approach using the worst-case response time (WCRT) of the update transaction such that the validity of the data object is guaranteed by completing the jobs of the update transaction before their deadlines.

Although the resulting update workload from *ML* has been shown to be lower than that obtained from *HH* (Ho et al., 1997), the total update workload from *ML* can still be heavy as it uses the WCRT in determining the update periods. Although this may help the update transactions achieve better data quality, it can severely hurt the schedulability of the system. In addition, in *ML*, it is assumed that the application transactions are assigned lower priorities compared with the update transactions in order to eliminate the impact of their scheduling on the scheduling of update jobs. Thus, the application transactions may not be schedulable when the update workload is heavy.

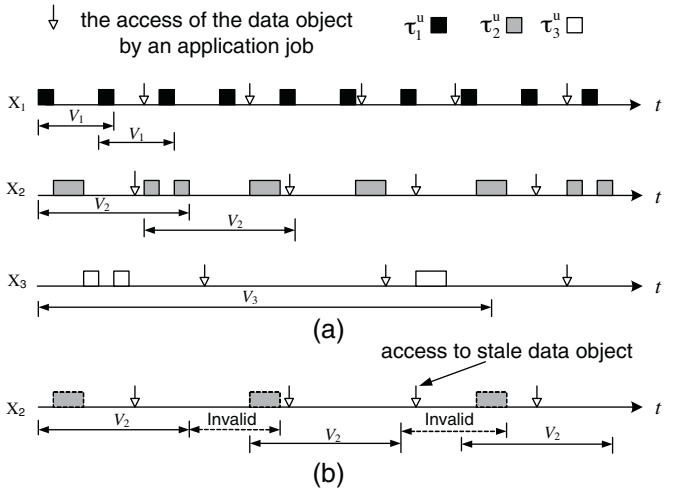

Fig. 2. The schedules of the update transactions in Example 3.1 before and after increasing the period of τ_2^u.

Example 3.1. Consider a simple example with three real-time data objects X_1, X_2 and X_3 to be refreshed by update transactions, τ_1^u, τ_2^u and τ_3^u, respectively. Fig. 2 shows a schedule obtained using *ML*. Following the assumption in *ML*, the application transactions are assigned lower priorities compared with the update transactions. In the example, we concentrate on the scheduling of update transactions to demonstrate the problems of *ML*.

Table 2 summarizes the periods and deadlines of τ_1^u, τ_2^u and τ_3^u calculated using *ML* (Xiong and Ramamritham, 2004). Although the total utilization for maintaining the validity of the three objects is $U = \sum_{i=1}^{3} C_i^u/P_i^u = 1/4 + 2/7 + 2/24 = 0.619$, the utilization for updating the three objects is very different. The utilization for X_3 is only 0.083 while that for X_1 is 0.25. It is three times of X_3. In addition, the update utilization for different data objects is independent on their probabilities to be accessed by the application transactions. As shown in Fig. 2, quite a number of update jobs for the data objects, (e.g., X_1) are wasted as no application job accesses to these data versions before they become invalid.

To reduce the update utilization for non-accessed data objects and the total update utilization, we may increase the validity intervals in calculating the periods for the update transactions. Fig. 2(b) shows the new update schedule for X_2 in which the period of transaction τ_2^u (denoted by $\tau_2^{u'}$) is increased from 7 to 14. Its utilization is reduced from 0.286 down to 0.143. As shown in Fig. 2(b) although X_2 may be stale for a certain period of time, the impact can be limited if it is not a hot object, e.g., only one application job accesses to a stale data object. Another important benefit of extending the validity periods is that if the set of update transactions is not schedulable, increasing the update periods may make it schedulable. In addition, reducing the update workload may also reduce its impact on the scheduling of the application jobs.

Table 2
Parameters of update transactions in Example 3.1.

Trans.	C_i^u	D_i^u	P_i^u	$\mathcal{V}_i$	Util.
τ_1^u	1	1	4	5	0.25
τ_2^u	2	3	7	10	0.286
τ_3^u	2	6	24	30	0.083
$\tau_2^{u'}$	2	3	14	10	0.143

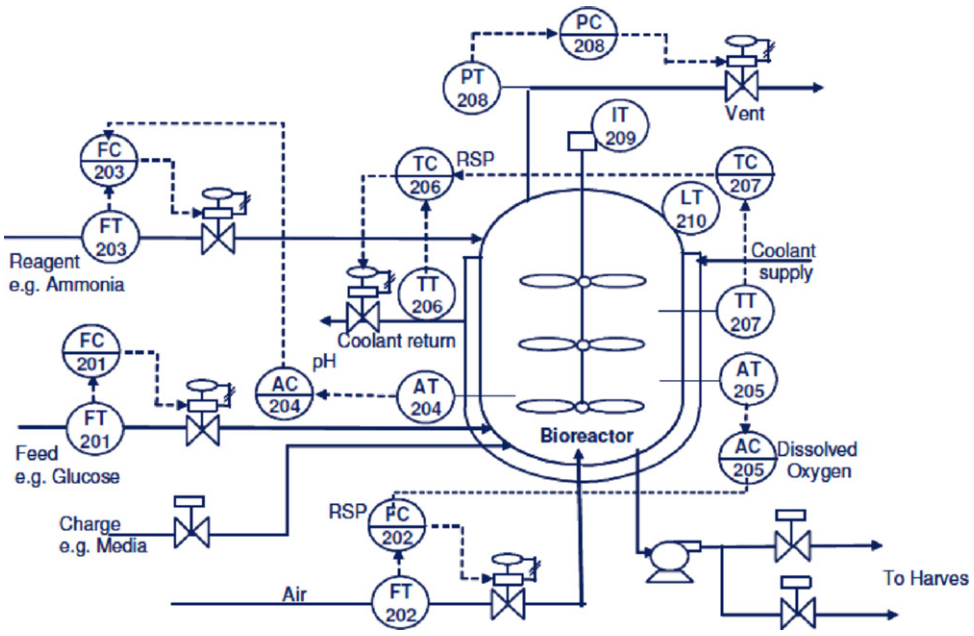

Fig. 3. A simple bio-reactor process with 10 transmitters, 5 regulating valves and 3 blocking valves.

4. An application example and system model

In this section, we first describe a simple bio-reactor process to illustrate the performance requirements of a real-time database system for surveillance of critical events. Following the application example, we then present the system model and transaction model used in this paper, and define the *quality of data* (*QoD*) for execution of application transactions.

4.1. An application example

As shown in Fig. 3, the bio-reactor process contains 10 transmitters (sensors), 5 regulating valves and 3 blocking valves. The details of the field devices are described in Nixon et al. (2008). Note that this bio-reactor process can be a part of a much larger scale industrial production system. For the reliability purpose, multiple sensors may be deployed for capturing the status of the same entity. Thus, the total number of sensors maintained in the system can be much larger.

The primary objective of the transmitters (sensors) is to provide real-time process measurements. Each sensor implements an update transaction for generating update jobs after capturing the latest status of the operation process. The frequency (or called the rate of update job generation) at which this information is required by the process automation host is specific to the process equipment and the measurement type, e.g., pressure, temperature, flow, level and analytical. The generated update jobs are submitted to a database server (not shown in the figure) for refreshing the corresponding real-time data objects maintained in the database.

The operation of each actuator is controlled by a periodic application transaction which accesses to a set of pre-defined data objects maintained in the real-time database according to the functions to be performed by the actuator. For example, the opening of a blocking valve depends on the values of the temperature sensors measuring the bio-reactor. When the actuator is a regulating valve, the automation host will periodically update the actuator setpoint as part of continuous control strategy according to the output from the corresponding application transaction. If the actuator is a blocking valve, then the automatic host will trigger a setpoint request only if the output from the corresponding application transaction satisfies a pre-defined threshold value. For this case, the setpoint will be updated on a change-of-state, e.g., the setpoint will only be written to the valve when the discrete setpoint changes. It is important to have a guarantee in meeting the deadlines of the application transactions. Otherwise, the status and safety in operation of the bio-reactor cannot be guaranteed. At the same time, to ensure that the decisions from the application transactions to the actuators are correct, it is also important to guarantee that the validity of all the sensor data are maintained above required levels for the execution of the application transactions. Otherwise, they may generate incorrect decisions to the actuators and the consequence can be catastrophic.

4.2. System model

Fig. 4 depicts the system model to be used in this paper. It represents a typical real-time database system for surveillance of critical events, including the bio-reactor process discussed in above. The system model consists of a fixed set of periodic application transactions and a set of update transactions. The set of application transactions is defined according to the surveillance requirements while the set of update transactions is defined according to the

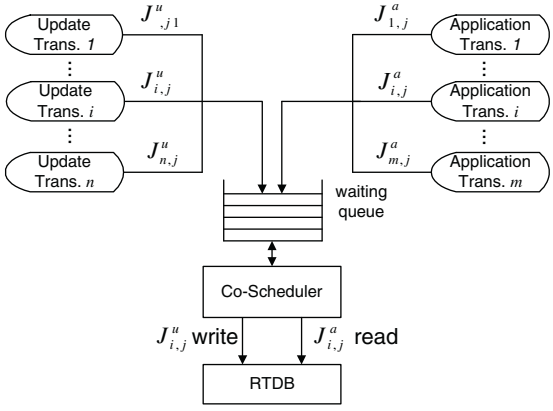

Fig. 4. An illustration of the system model.

number of real-time data objects maintained in the real-time database. Each invocation of an application transaction τ_i^a is called an application job $J_{i,j}^a$ whose deadline $d_{i,j}^a$ is defined to be its release time $r_{i,j}^a$ plus a fixed relative deadline D_i^a. Each update transaction τ_i^u generates an update job $J_{i,j}^u$ at the beginning of every period P_i^u to refresh data object X_i with a validity interval $\mathcal{V}_i$. Unlike *ML*, as will be explained in Section 5, the period for generating update jobs in an update transaction may not be fixed in *AEDF-Co* with the purpose to reduce the update workload. After installing an update job τ_i^a for X_i, the new version of X_i is valid up to time $r_{i,j}^a + \mathcal{V}_i$. The released update jobs and application jobs are sorted according to their priorities in the waiting queue as shown in Fig. 4. The co-scheduler selects the highest priority job from the waiting queue for processing and the scheduling is preemptive. If a higher priority job is released, i.e., a higher priority job is inserted into the waiting queue, the job that is processing will be preempted and returned into the waiting queue. The main goals of our approach are to determine how to generate update jobs from update transactions and schedule both update and application jobs in the waiting queue for execution, such that the deadline constraints of all the application transactions are satisfied and at the same time, the quality of data (QoD) of the real-time data objects is maximized.

4.3. Update transaction model and assumptions

In our update transaction model, each update transaction τ_i^u is defined as a 4-tuple $\tau_i^u = (C_i^u, \mathcal{V}_i^{min}, \mathcal{V}_i^{max}, O_i)$, where C_i^u is the worst-case execution time (WCET) for installing an update job of τ_i^u into the real-time database system; $\mathcal{V}_i^{min}$ is the minimum validity interval and $\mathcal{V}_i^{max}$ is the maximum validity interval of τ_i^u. The validity interval $\mathcal{V}_i$ for an update transaction τ_i^u to maintain the validity of data object X_i is then determined within this range, i.e., $\mathcal{V}_{min} \le \mathcal{V}_i \le \mathcal{V}_{max}$. The details of the definitions for $\mathcal{V}_i^{min}$ and $\mathcal{V}_i^{max}$ will be discussed in Section 4.5. O_i defines the set of application transactions that will access to X_i.

Since different update transactions refresh different data objects and each update job just updates one data object, no concurrency control is considered in processing update jobs (Ramamritham et al., 2004). To further simplify the problem, it is assumed that the real-time data objects are maintained in main memory such that no I/O access delay is required in accessing a data object.

4.4. Application transaction model and assumptions

In our application transaction model, each application transaction is defined as a 4-tuple $\tau_i^a = (C_i^a, D_i^a, P_i^a, \Omega_i)$, where C_i^a is the worst-case execution time (WCET) for executing an application job of τ_i^a; D_i^a is the relative deadline; P_i^a is the period for generating application jobs; and Ω_i is the set of update transactions which provide new data values for τ_i^a, that is $\Omega_i = \{< \tau_k^u, \alpha_{i,k} >\}_{k=1}^{|\Omega_i|}$ where $\alpha_{i,k}$ is the importance factor of τ_k^u to τ_i^a and $\sum_k \alpha_{i,k} = 1$. Different data objects may have different importance to the function to be performed by an application transaction and a larger value of $\alpha_{i,k}$ means that data object X_k is of higher importance to it.

Each invocation of an application transaction is called an application job which consists of a set of pre-defined operations. To simplify the discussion, the operations are assumed to be executed one by one, and each operation reads one real-time data object. Based on the values of the accessed data objects, it generates a decision according to the defined surveillance function to respond to the events occurring in the operation environment.

Since the function of each application transaction to be performed in a system are pre-defined according to system requirements, the period, the deadline constraint, the data requirements and the processing time for each application transaction are pre-defined or can be pre-analyzed.

4.5. Quality of data (QoD) and quality of service (QoS)

Observing stale data objects can make an application job generate incorrect response in event surveillance. Thus, an important performance objective in addition to meeting the deadlines of all the application jobs is to maximize the *quality of data (QoD)* observed by an application job.

In Ramamritham (1993), it uses a validity interval to define the quality of a data object. This method is simple but may not be sufficient for the systems which need to co-schedule update and application transactions. In digital real-time surveillance and control systems, sensors are usually configured with a range of sampling frequencies for defining the quality of real-time data objects to improve the system flexibility (Seto et al., 1996; HART, 2007). In this paper, following Seto et al. (1996), we propose to use a range of validity intervals, $[\mathcal{V}_i^{min}, \mathcal{V}_i^{max}]$, for each data object X_i and its corresponding update transaction τ_i^u as introduced in the previous subsection. The values of $\mathcal{V}_i^{min}$ and $\mathcal{V}_i^{max}$ of a data object are system-dependent. The definition of $\mathcal{V}_i^{min}$ is the same as the original definition of validity interval in Ramamritham (1993) while the value for $\mathcal{V}_i^{max}$ of a data object can be defined based on the dynamic property of the corresponding entity in the operation environment. If the dynamic property of an entity is higher, the difference in value between the $\mathcal{V}_i^{min}$ and $\mathcal{V}_i^{max}$ will be smaller. After $\mathcal{V}_i^{min}$, although a data object is considered to be stale, it still has some "value" to the application jobs. A new update job $J_{i,j}^u$ released from τ_i^u at time $r_{i,j}^u$ has to be installed into the database before $r_{i,j}^u + \mathcal{V}_i^{max}$; otherwise, the quality of the data object provided by completing $J_{i,j}^u$ will be considered to be too poor to be used by any application jobs.

The main benefit of using a range of validity intervals is that it provides flexibility in scheduling for tradeoff between the quality of data objects and the system schedulability. In cases that we cannot meet the deadlines of all the application jobs while ensuring that all the data objects are valid as defined by their minimum validity intervals, we may use larger validity intervals to schedule them provided that the validity intervals are not larger than the maximum validity intervals of the update transactions (Seto et al., 1996; HART, 2007). For these cases, a data object may be invalid for a certain period of time but the maximum degree of staleness is bounded by the maximum validity intervals of the update transactions.

Different surveillance applications may have different functions for quantifying the quality of a data object as a function of time after $\mathcal{V}_i^{min}$. In this paper, to simplify the discussion, we follow the definition of staleness (Definition 3.2) to define the *quality of data* (*QoD*) of data object X_i at time t:

$$\mathcal{Q}_i^u(t) = 1 - \frac{S_i(t)}{\mathcal{V}_i^{max} - \mathcal{V}_i^{min}} \tag{1}$$

where $S_i(t)$ is the staleness of X_i at time t defined according to Definition 3.2. According to Eq. (1), the QoD of X_i is bounded by 1 because the staleness is not smaller than 0. The QoD of a data object decreases with an increase in data staleness. It is assumed that the impact of data staleness to the QoD of a data object depends on the difference between the minimum and maximum validity intervals of the data object. The minimum validity interval gives the best QoS (i.e., QoD = 1) while the QoD of the data object reduces to zero when its validity is extended to the maximum validity interval. If the QoD of X_i is between 0 and 1, then X_i still has some "value" to the application jobs. Once the staleness becomes larger than $\mathcal{V}_i^{max} - \mathcal{V}_i^{min}$, the QoD of X_i will become negative. A negative value of QoD of a data object indicates that the data object is useless or even harmful to the application jobs. As shown in Fig. 5, we assume that an update

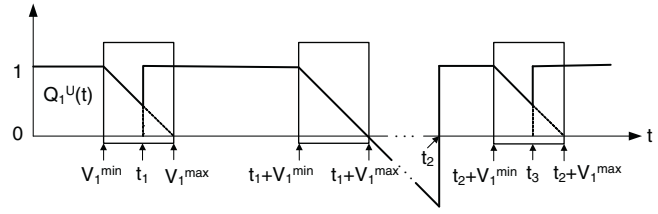

Fig. 5. An example of the quality of data (QoD) of data object X_1.

job of τ_i^u is released at time 0. From time $\mathcal{V}_i^{min}$, the QoD of X_i starts to decrease linearly until a new update job is installed at time t_1. Then, the QoD of X_i is restored to 1 and then decreases again starting at time $t_1 + \mathcal{V}_i^{min}$.

Based on the definition of the QoD for a data object, we can define the *quality of service* (*QoS*) obtained from completing an application job. We use $\mathcal{Q}_i^a(t)$ to denote the *QoS* obtained from completing τ_i^a as a function of time. The impact of a stale data object on an application job is application dependent. Herein, it is assumed that $\mathcal{Q}_i^a(t)$ is the sum of the *QoD* $\mathcal{Q}_k^u(t)$ ($\tau_k \in \Omega_i$) at the time when τ_i^a is finished, weighted by the importance factor $\alpha_{i,k}$. This definition is suitable for aggregate surveillance queries which access to a set of data objects to derive a result from the set of data objects, e.g., an application job calculates the mean temperature of a set of neighboring temperature sensors in process management.

$$\mathcal{Q}_i^a(t) = \begin{cases} \sum_{k=1}^{|\Omega_i|} \alpha_{i,k} \cdot \mathcal{Q}_k^u(t) & \mathcal{Q}_k^u(t) \geq 0 \\ 0, & \text{o.w.} \end{cases} \quad (2)$$

Note that if an application job misses its deadline, the *QoS* obtained from it will be reduced to zero. Thus, to ensure the correctness of the surveillance functions, it is important to meet the deadlines of the application jobs as well as to maximize the *QoD* by timely completion of update jobs.

5. Adaptive co-scheduling: *AEDF-Co*

In this section, we first describe the baseline methods, the *Update First* (*UF*), *Application First* (*AF*) and the Interleaved U/A schemes for co-scheduling the two types of transactions, and their limitations. Then, we introduce a novel co-scheduling algorithm called *Adaptive Earliest Deadline First Co-Scheduling* (*AEDF-Co*) to resolve the real-time co-scheduling problem. Its performance goal is to find a schedule for given sets of application and update transactions such that all the application transactions can meet their deadline constraints while the QoD of the data objects is maximized.

5.1. Baseline algorithms and their limitations

The update and application transaction co-scheduling problem basically consists of two sub-problems: (1) how to generate update jobs from an update transaction to maintain the data validity; and (2) how to determine the priorities for the update and application jobs so that all of them can be completed before the deadlines.

Two baseline solutions to address this co-scheduling problem are the *Update First* (*UF*) and *Application First* (*AF*) methods. *UF* and *AF* represent two extremes in co-scheduling: *UF* assigns higher priorities to update transactions to maximize the QoD of the real-time data objects while *AF* assigns higher priorities to the application transactions to minimize the probability of violating their deadline constraints.

To maintain the data validity, both *UF* and *AF* apply the *More–Less* (*ML*) approach to determine the periods and deadlines of the update transactions using a fixed validity interval of a data object, and then they use *Deadline Monotonic* (*DM*) to schedule the update jobs. In *UF* and *AF*, the application jobs are also scheduled using *DM* according to their relative deadlines. Although both the application and update transactions use *DM* to prioritize the jobs, we cannot simply schedule them together. This is because *ML* does not consider the possible preemption from the application transactions when calculating the periods and deadlines for the update transactions. Interleaving the executions of the update and application jobs may make some of them miss the deadlines and make the overall performance totally unpredictable. *UF* and *AF* simplify this problem by either assigning higher priorities to the update transactions or application transactions, respectively. In this way, they can either maximize the QoD obtained from the update transactions or maximize the schedulability of the application transactions. These simple solutions, however, inevitably will either reduce the schedulability of the application transactions (i.e., in *UF*) or the QoD (i.e., in *AF*) especially when the system workload is heavy. One simple way to balance the QoD and meeting the deadlines of application jobs is to interleave the priorities of *AF* and *UF* according to a fixed ratio. We call this scheme as the *Interleaved-U/A* scheme in which a certain percentage, called the *relative priority factor*, of update transactions are assigned to higher priorities relative to the set of application transactions. For example if the value of the relative priority factor is 0.1, 10 percents of update transactions are assigned to lower priorities relative to the set of application transactions. Of course, the biggest performance problem of the *Interleaved-U/A* scheme is how to set the value for the relative priority factor to maximize the overall system performance.

5.2. The design principles of AEDF-Co

To overcome the drawbacks of *UF* and *AF* and to maximize the performance of both application and update transactions, it is important to dynamically adjust the relative priorities of the application and update jobs in the co-scheduling. In this section, we present the design principles of the *Adaptive Earliest Deadline First Co-Scheduling* (*AEDF-Co*) algorithm which has four important features.

Firstly, we introduce a deferrable scheduling (Xiong et al., 2008a) into *AEDF-Co*. *ML* is pessimistic on the deadline and period assignments for an update transaction as it uses the worst-case response time of an update transaction to determine the period and relative deadline of its jobs. The pessimistically calculated relative deadline D_i^u will derive a relatively short period P_i^u ($P_i^u = \mathcal{V}_i - D_i^u$) especially for the update transactions with shorter validity intervals, and impose a heavy update workload on the system. This can severely hurt the system schedulability and degrade the performance of the application transactions if the update transactions are assigned higher priorities as in *UF*. To resolve this problem, *AEDF-Co* defers the release time of an update job as late as possible if the quality of the corresponding data object will not be affected, i.e., the next job is completed before the previous version of the corresponding data object becomes invalid. In *AEDF-Co*, the deadline $d_{i,k}^u$ of $J_{i,k}^u$ is set as the release time of the previous job $J_{i,k-1}^u$ plus $\mathcal{V}_i$ which is set to be the value of its minimum validity interval, i.e., $r_{i,k-1}^u + \mathcal{V}_i^{min}$. After setting the deadline of $J_{i,k}^u$, its release time $r_{i,k}^u$ is calculated backwards from its deadline $d_{i,k}^u$ by taking its execution time C_i^u and the total preemption from higher priority jobs (both update and application jobs) into consideration. As shown in Fig. 6(a), compared with the one obtained from *ML*, the release time of $J_{i,k}^u$ is deferred from $r_{i,k}^u$ to $r_{i,k}^{u'}$. The deferral of the release time reduces the relative deadline of the update job $J_{i,k}^u$ from $d_{i,k}^u - r_{i,k}^u$ to $d_{i,k}^u - r_{i,k}^{u'}$ and increases the separation between $J_{i,k-1}^u$ and $J_{i,k}^u$. This will help reduce the utilization of the update transactions and allow

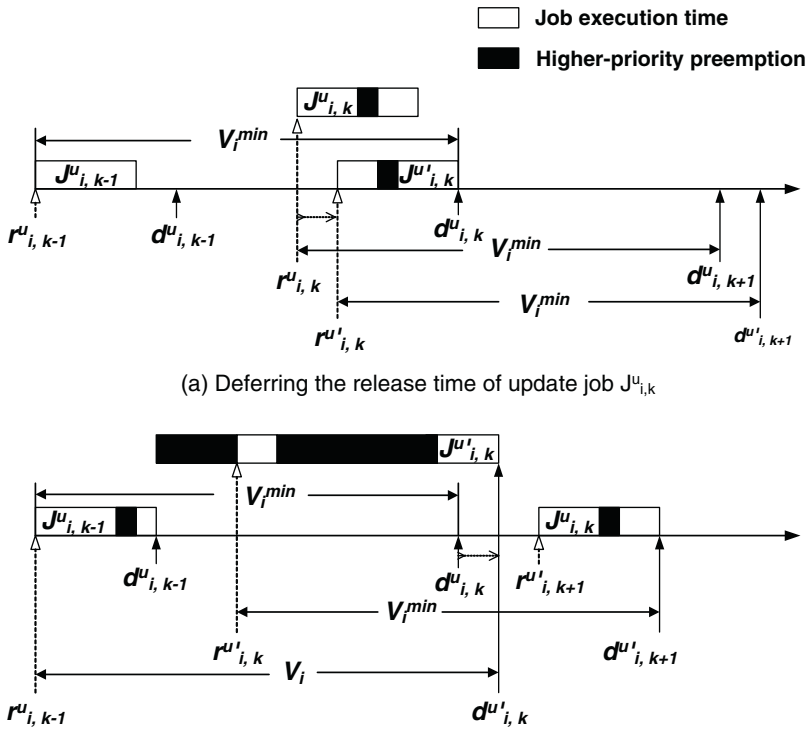

Fig. 6. An illustration of calculating the release time $r^u_{i,k}$ for update job $J^u_{i,k}$ in *AEDF-Co*.

more application transactions to be scheduled in the system. Thus, unlike *ML*, the update job generations in *AEDF-Co* may not follow a fixed period.

The second important feature of *AEDF-Co* is to use a range of validity intervals ($\mathcal{V}_i^{min}$ and $\mathcal{V}_i^{max}$) to determine the validity intervals $\mathcal{V}_i$s for the update jobs of an update transaction, τ_i^u such that different jobs of the same update transaction may have different validity intervals. Initially, we set $\mathcal{V}_i$ to be $\mathcal{V}_i^{min}$ with the purpose to maximize the QoD of data object X_i. In cases where an update job is not schedulable, a larger value of $\mathcal{V}_i$ will be used provided that it is not larger than $\mathcal{V}_i^{max}$. For example as shown in Fig. 6(b), if the deadline $d^u_{i,k}$ of $J^u_{i,k}$ is set to be $r^u_{i,k-1} + \mathcal{V}_i^{min}$, $J^u_{i,k}$ is not schedulable as the derived release time $r^u_{i,k}$ is smaller than the deadline $d^u_{i,k-1}$ of $J^u_{i,k-1}$ due to preemption from higher-priority jobs (both update and application jobs). To solve the problem, the deadline $d^u_{i,k}$ of $J^u_{i,k}$ is increased from $r^u_{i,k-1} + \mathcal{V}_i^{min}$ up to $r^u_{i,k-1} + \mathcal{V}_i^{max}$ such that $J^u_{i,k}$ is schedulable. As shown in Fig. 6(b), the new deadline and release time for $J^u_{i,k}$ are $d^{u'}_{i,k}$ and $r^{u'}_{i,k}$, respectively, after extending the value for V_i.

Thirdly, as illustrated in Example 3.1, some update jobs are unnecessary and wasted as the new versions created by them are never accessed by any application jobs. In order to minimize the number of such update jobs, we include a *look-ahead* policy into *AEDF-Co*. If the release time of an update job is to be determined, *AEDF-Co* will first look ahead the waiting queue and find the earliest release time t among all the application jobs which will access to the data object to be updated by this update job. If the deadline of the update job is earlier than t, then it will be deferred to t. Then, the release time of the update job will be calculated based on the new deferred deadline. For example, as shown in Fig. 7, the deadline of update job $J^u_{i,k}$ is first deferred to $d^{u'}_{i,k}$. By observing that the earliest release time of the application jobs that will access to X_i is $r^a_{1,j}$, the deadline $d^u_{i,k}$ is further deferred from $d^{u'}_{i,k}$ to $r^a_{1,j}$, and then the release time $r^u_{i,k}$ is deferred accordingly. In this way, the look-ahead policy will further reduce the update workload in the system.

Finally, we adopt the dynamic scheduling, the earliest deadline first (EDF), to schedule the application jobs and update jobs based on their deadlines. It is expected that the dynamic scheduling will be more adaptive to the urgencies of the jobs such that the overall schedulability of the system can be improved.

5.3. The algorithm

Algorithm 1 summarizes the framework of the *AEDF-Co* algorithm. In Algorithm 1, it is assumed that all the update and application transactions are synchronized at time 0 and all the real-time data objects are temporally valid initially. The waiting queue Q_{EDF} stores the update and application jobs in the ascending order of their deadlines, i.e., using *EDF*. Initially, the first application job $J^a_{i,0}$ of each application transaction τ_i^a and first job $J^u_{i,0}$ of each update transaction τ_i^u with deadline $\mathcal{V}_i^{min}$ are put into Q_{EDF} (Lines 1 and 2).

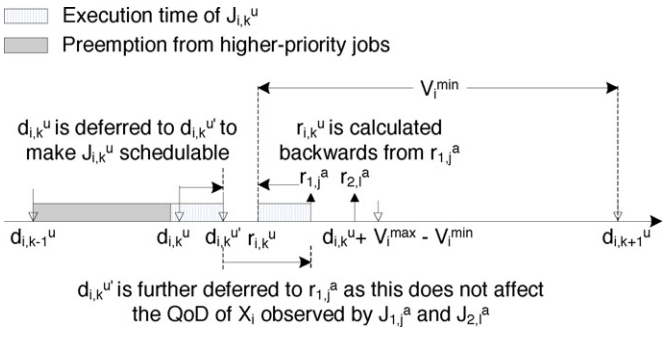

Fig. 7. An illustration of adjusting the deadline of an update job.

Algorithm 1 *(Adaptive Earliest Deadline First Co-Scheduling).*

Input: Update transactions $\mathcal{T}^u = \{\tau_i^u\}_{i=1}^m$ and application transactions $\mathcal{T}^a = \{\tau_i^a\}_{i=1}^n$
Output: Construct a schedule $\mathcal{S}$ if $\mathcal{T}^u$ and $\mathcal{T}^a$ are all schedulable; otherwise, reject.
1: Insert all 1st jobs $J_{i,0}^u$ of τ_i^u in Q_{EDF} by setting $d_{i,0}^u = \mathcal{V}_i^{min}$.
2: Insert all 1st jobs $J_{i,0}^a$ of τ_i^a in Q_{EDF}.
3: **while** TRUE **do**
4: **if** the first job $J_{i,k}$ from Q_{EDF} is an application job or an update job with determined release time **then**
5: Schedule $J_{i,k}$ using *EDF*;
6: **if** $J_{i,k}$ is finished **then**
7: $d_{i,k+1} = r_{i,k} + \mathcal{V}_i^{min}$ if $J_{i,k}$ is an update job;
8: $d_{i,k+1} = r_{i,k} + P_i^a + D_i^a$ if $J_{i,k}$ is an application job;
9: Enqueue $J_{i,k+1}$ to Q_{EDF};
10: **end if**
11: **else**
12: Dequeue $J_{i,k}$ from Q_{EDF};
13: $d_{i,k}$ = **AdjustUpdateDeadline**($J_{i,k}$);
14: **if** $d_{i,k} \leq$ the deadline of the 1st job in Q_{EDF} **then**
15: **if** $k == 0$ **then**
16: $r_{i,k}$ = **CalcUpdateReleaseTime**(i, k, 0, $d_{i,k}$);
17: **else**
18: $r_{i,k}$ = **CalcUpdateReleaseTime**(i, k, $d_{i,k-1}$, $d_{i,k}$);
19: **end if**
20: **else**
21: Enqueue $J_{i,k}$ to Q_{EDF};
22: **end if**
23: **end if**
24: **end while**

After the initialization, we dequeue the first job $J_{i,k}$ from Q_{EDF}. If $J_{i,k}$ is an application job or an update job whose release time has been derived (Lines 4–10), we will schedule them using *EDF* and insert the next job of the same transaction into the Q_{EDF} if the current job is finished. If $J_{i,k}$ is an update job whose release time has not been derived (Lines 11–23), we will first perform the look-ahead policy by invoking Algorithm 3 to defer its deadline $d_{i,k}^u$ as much as possible as long as the QoD of the corresponding data object to be accessed by the following application jobs in Q_{EDF} will not be affected. Algorithm 3 first searches a deadline d that can make job $J_{i,k}^u$ schedulable (Lines 2–11 of Algorithm 3). Then it searches the earliest release time r of the application jobs that will access to the data object to be updated by $J_{i,k}^u$ (Lines 13–18 of Algorithm 3). The maximum of d and r is chosen to be the deadline of $J_{i,k}^u$ (Line 19 of Algorithm 3). After the adjustment, if $d_{i,k}^u$ is not the earliest deadline in Q_{EDF}, $J_{i,k}^u$ will be re-inserted into Q_{EDF}. Then, we will dequeue the first job in Q_{EDF} and repeat the same process; Otherwise, we calculate its release time $r_{i,k}^u$ backwards from $d_{i,k}^u$ by calling Algorithm 2. It searches backwards from the deadline $d_{i,k}^u$ of $J_{i,k}^u$ to the deadline $d_{i,k-1}^u$ of the previous job $J_{i,k-1}^u$ and allocates the idle slots that are not occupied by any higher-priority jobs to $J_{i,k}^u$ until the remaining execution time of $J_{i,k}^u$ is 0. Once the remaining execution time of $J_{i,k}^u$ is reduced to 0, the time point is chosen to be the release time $r_{i,k}^u$ for $J_{i,k}^u$.

Algorithm 2 *(CalcUpdateReleaseTime(i, k, $d_{i,k-1}$, $d_{i,k}$)).*

Input: $J_{i,k}$ and time interval $[d_{i,k-1}, d_{i,k})$.
Output: Release time $r_{i,k}$.
1: $C_R = C_i$; //C_R is the remaining execution time of $J_{i,k}^u$.
2: $r_{i,k} = d_{i,k}$;
3: **while** $r_{i,k} \geq d_{i,k-1}$ **do**
4: **if** time slot $r_{i,k}$ is not scheduled **then**
5: Schedule time slot $r_{i,k}$ for $J_{i,k}^u$;
6: $C_R - -$; //allocate the idle slot to $J_{i,k}^u$ and the remaining execution time decreases by 1.
7: **end if**
8: **if** $C_R == 0$ **then**
9: **return** $r_{i,k}$; //get the release time of $J_{i,k}^u$ when C_R is 0.
10: **end if**
11: **end while**
12: **return** FAILURE; //the release time is smaller than $d_{i,k-1}$.

Table 3
Parameters of update and application transactions in Example 5.1.

	C_i^u / C_i^a	$\mathcal{V}_i^{min}$	$\mathcal{V}_i^{max}$	D_i^u / D_i^a	P_i^u / P_i^a
τ_1^u	1	5	9	1	4
τ_2^u	2	8	12	3	5
τ_1^a	1	–	–	5	8
τ_2^a	2	–	–	6	9

Algorithm 3 *(AdjustUpdateDeadline($J_{i,k}^u$, $d_{i,k}$)).*

Input: The update job $J_{i,k}^u$ and its original deadline $d_{i,k}$.
Output: Adjusted deadline $d'_{i,k}$.
1: //search the smallest deadline for $J_{i,k}^u$.
2: **for** $d = d_{i,k}$ to $d_{i,k} + \mathcal{V}_i^{max} - \mathcal{V}_i^{min}$ **do**
3: **if** $k == 0$ **then**
4: $r_{i,k}$ = **CalcUpdateReleaseTime**(i, k, 0, d); //$J_{i,k}^u$ is the first job of τ_i^u.
5: **else**
6: $r_{i,k}$ = **CalcUpdateReleaseTime**(i, k, $d_{i,k-1}$, d); //$J_{i,k}^u$ is not the first job of τ_i^u.
7: **end if**
8: **if** $r_{i,k} \neq$ FAILURE **then**
9: **break**; //find the earliest deadline.
10: **end if**
11: **end for**
12: //search the earliest release time of the application jobs that will access to the data object to be updated by $J_{i,k}^u$.
13: $r = d_{i,k} + \mathcal{V}_i^{max} - \mathcal{V}_i^{min}$;
14: **for** each $\tau_k^a \in O_i$ **do**
15: **if** the earliest job of τ_k^a in Q_{EDF} is $\tau_{k,l}^a$ **then**
16: $r = \min\{r, r_{k,l}^a\}$;
17: **end if**
18: **end for**
19: **return** $d'_{i,k} = max\{d, r\}$; //choose the larger one as the deadline of $J_{i,k}^u$.

Example 5.1. Consider a set of two update transactions $\{\tau_1^u, \tau_2^u\}$ and a set of two application transactions $\{\tau_1^a, \tau_2^a\}$ with parameters shown in Table 3. It is assumed that application transaction τ_1^a will access to data objects X_1 and then X_2 sequentially, and τ_2^a will access to data object X_2 only. The validity of X_1 and X_2 is maintained by τ_1^u and τ_2^u, respectively.

In *UF* and *AF*, the periods and relative deadlines of τ_1^u and τ_2^u are determined by *ML* using the minimum validity intervals of X_1 and X_2. The schedules obtained from *UF* and *AF* are shown in Fig. 8(a) and (b), respectively. As shown in Fig. 8(a) and (b), both *UF* and *AF* cannot schedule the transaction sets successfully. In *UF*, application job $J_{1,0}^a$ misses its deadline while in *AF* both $J_{1,0}^u$ and $J_{2,0}^u$ miss their deadlines.

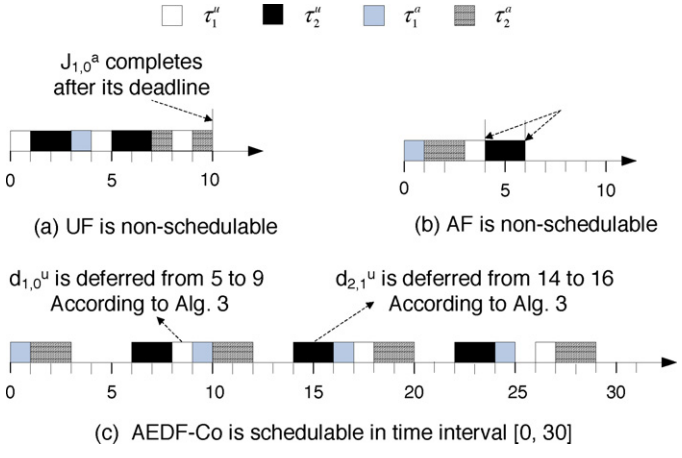

Fig. 8. *AEDF-Co* outperforms *UF* and *AF* in scheduling the jobs.

Table 4
Parameter settings in the experiments.

Parameter	Value	Parameter	Value
# of Update Trans.	[50, 375]	# of App. Trans.	[30, 80]
C_i^u	[5, 10]	C_i^a	[10, 20]
$\mathcal{V}_i^{min}$	[2500, 10,000]	P_i^a	[200, 8000]
$\mathcal{V}_i^{max}$	$1.5 \times \mathcal{V}_i^{min}$	D_i^a	$0.4 \times P_i^a$

Fig. 8(c) shows the schedule obtained from *AEDF-Co*. At initial time 0, both $J_{1,0}^u$ and $J_{1,0}^a$ have the same earliest deadline at time 5. It is assumed that $J_{1,0}^a$ is selected to execute. Job $J_{1,0}^a$ is finished at time 1. Then, $J_{1,0}^u$ is the job with the earliest deadline. According to the deferrable and look-ahead policies in Algorithm 3, its deadline is deferred to time 9. Therefore, application job $J_{2,0}^a$ becomes the job with the earliest deadline and is selected for execution. At time 3, no job has been released in Q_{EDF} and the release time of the update job with the earliest deadline will be calculated. Then, $J_{2,0}^u$ has the earliest deadline. Its release time $r_{2,0}^u$ is derived to be at time 6 and the next job $J_{2,1}^u$, which has a deadline at time 14, i.e., $r_{2,0}^u + 8$, is inserted into Q_{EDF}. Similarly, the release time $r_{1,0}^u$ of $J_{1,0}^u$ is derived to be at time 8, and the next job $J_{1,1}^u$ which has a deadline at time 13, i.e., $r_{1,0}^u + 5$, is inserted into Q_{EDF}.

6. Experiment evaluation and discussions

In this section, we present the important results obtained from our performance studies on *AEDF-Co*. In the experiments, we compare *AEDF-Co* with the *Update first* (*UF*) and *Application First* (*AF*) methods as well as the combined scheme, Interleaved-U/A using different relative priority factors. Since the validity interval in *AEDF-Co* is chosen within a range, $[\mathcal{V}_i^{min}, \mathcal{V}_i^{max}]$ while a single fixed value of validity interval is used for each update transaction in *ML*, for better comparison, in *UF*, *AF* and Interleaved-U/A, we repeat the experiments using the $\mathcal{V}_i^{min}$ and $\mathcal{V}_i^{max}$ to be the fixed validity value for update transaction τ_i^u.

6.1. Simulation model, parameters and measures

The simulation model is developed according to the system model introduced in Section 4.2. It consists of a real-time controller and a fixed set of sensor nodes. Each sensor node runs an update transaction to generate update jobs following the release times determined by the adopted method, i.e., *ML* or *AEDF-Co*. At the same time, a set of application transactions is maintained at the controller to generate application jobs following the pre-defined periods to access to the data objects maintained in the real-time database. In the simulation model, we do not consider the concurrency control between the update transactions and application transactions as the conflicts can easily be resolved by the application of the concept of data similarity (Kuo and Mok, 1993). It is also assumed that the synchronization delay in accessing shared data object is small and can be ignored as all the data objects are resided in the main memory and the problem of priority inversion can be resolved by the use of the priority inheritance method (Sha et al., 1990).

Table 4 summarizes the set of parameters and their default settings used in the experiments. They are determined according to the parameter settings used in previous studies (Xiong and Ramamritham, 2004) with some modifications to fit them into our system model.

Since one of the performance goals is to meet the deadline constraints of all the application transactions, one of the important performance metrics used in the experiments is the miss rate of application jobs (MR_A) which is defined as the number of application jobs which miss their deadlines over the total number of application jobs generated. Another important performance metric is NR_{QoD}, which is defined as the number of completed application jobs that have accessed to at least one data object with negative quality over the total number of application jobs completed. If an application job accesses to a data object with negative quality, the result generated from it is unpredictable and can be incorrect. In addition, we also measure the rate of update jobs that miss their deadlines (MR_U), the average QoS ($\overline{QoS}$) obtained from the application jobs as defined from Eq. (2), the average QoD of each data object ($\overline{QoD}$), the average QoD of all data objects ($\overline{QoD}_{all}$), and the worst-case QoD of each data object ($\overline{QoD}_w$). The $\overline{QoD}$ of a data object is the average of the lowest *QoD* of each version of the data object. If the $\overline{QoD}$ of a data object is lower, the probability of accessing a data object with negative quality will be higher. The $\overline{QoD}_{all}$ is the average of $\overline{QoD}$ of all the data objects. The $\overline{QoD}_w$ of a data object is the lowest *QoD* of the data object observed by the application jobs. Similar to NR_{QoD}, it is another performance indicator of the QoD observed by the application jobs.

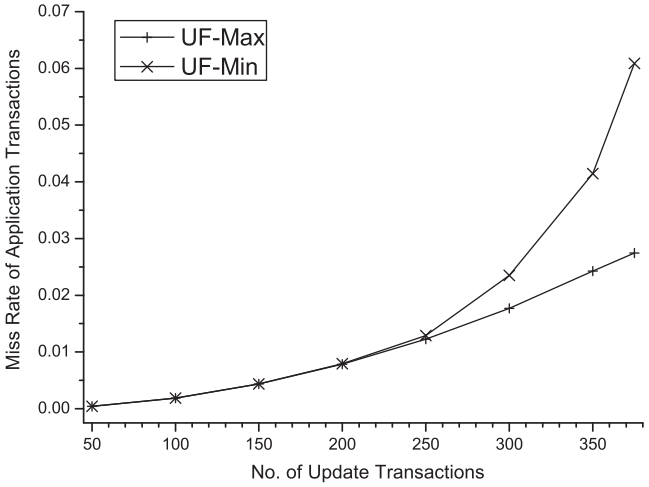

Fig. 9. MR_A vs. # of Update Trans.

6.2. Experimental results

We have conducted extensive experiments to evaluate the performance of *AEDF-Co*. For the limit of space, three representative experiments are presented in the following.

6.2.1. Impact of update workload

In the first set of experiments, we fix the number of application transactions at 60 and vary the number of update transactions from 100 to 375 to investigate the performance of *AEDF-Co* as compared with *UF* and *AF* under different update workloads. Increasing the number of update transactions increases the total update workload in the system. Fig. 9 depicts the miss rate of application jobs (MR_A) when the number of update transactions is varied. It is important to observe that only the MR_A of *UF* is greater than zero and its MR_A increases consistently with the number of update transactions. This is because in *UF* higher priorities are assigned to the update transactions. Increasing the workload of the update transactions increases the preemption from update jobs to the application jobs, and thus increases the probability of missing deadlines of the application jobs. As shown in Fig. 9, the MR_A of *UF*-min is consistently higher than that of *UF*-max. It is because with the use of smaller validity intervals in *UF*-min, the periods of the update transactions are shorter and the total workload of the update transactions is higher. Thus, the preemption of the application jobs from the update jobs is longer. The MR_A of *AF* remains zero even when the update workload

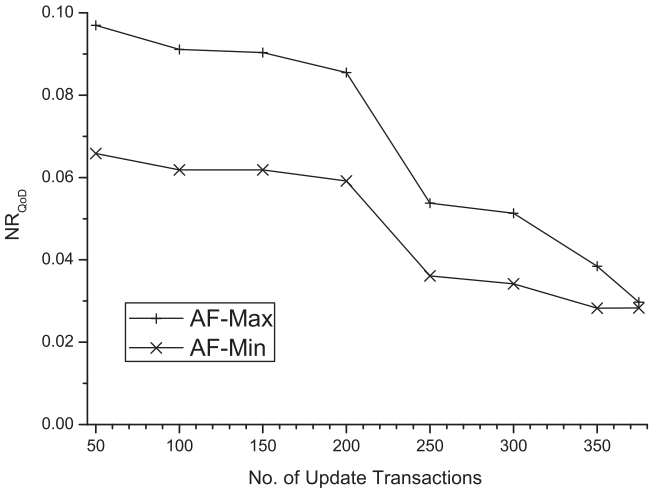

Fig. 10. NR_{QoD} vs. # of Update Trans.

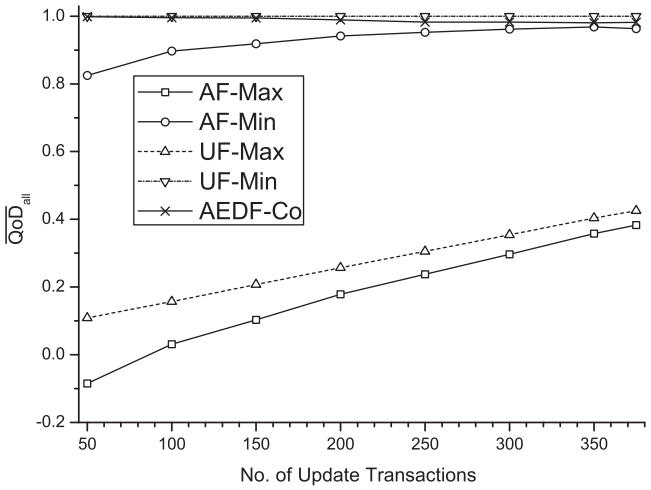

Fig. 12. $\overline{QoD}_{all}$ vs. # of Update Trans.

is very heavy. This is because, in *AF*, higher priorities are assigned to the application transactions. Thus, the performance of the application jobs is not affected by the update workload and all of them can be finished before their deadlines if the total workload of the application transactions is not very heavy. The MR_A of *AEDF-Co* is also remained zero for different update workloads. This illustrates that *AEDF-Co* is effective in meeting the deadlines of the application jobs even when the update workload is heavy.

In addition to meeting the deadlines of the application jobs, *AEDF-Co* is also effective in maintaining the QoD for the execution of the application jobs. As shown in Fig. 10, the NR_{QoD} of *AEDF-Co* also remains zero even when the update workload is heavy. On the other hand, as shown in Fig. 10, although in *AF* all the application jobs can be completed before the deadlines, the NR_{QoD} of *AF* is consistently higher than zero for different number of update transactions. It reaches a value of about 10% when the number of update transactions equals to 50. Thus, although all the application jobs can be completed before their deadlines, the results generated by them may be incorrect. The higher NR_{QoD} of *AF* is because some update jobs miss their deadlines as depicted in Fig. 11. In *AF*, lower priorities are assigned to the update jobs compared with the application jobs. Thus, they may miss their deadlines if the preemption from the application jobs is long. Missing the deadline of an update job can seriously degrade the quality of the corresponding data object and make the probability of accessing a data object with negative quality much higher. It is worth to note that the MR_U of *AF* decreases with an increase in number of update transactions. This is because in *AF* the worst-case response time of an update transaction is used for calculating its relative deadline. Thus, if the number of update transactions is larger, the deadline constraints of the lower-priority transactions will be looser. Therefore, they have lower probabilities of missing deadlines due to preemption from application jobs.

Consistent with the results shown in Fig. 10, the average QoD of all the data objects ($\overline{QoD}_{all}$) of *AF* is in general worse compared with *UF* and *AEDF-Co* as depicted in Fig. 12. As shown in Fig. 12, the $\overline{QoD}_{all}$ of *UF*-min remains close to 1 for different number of update transactions as it assigns higher priorities to the update transactions and uses smaller validity intervals. It is also interesting to observe that similar to *UF*-min, the $\overline{QoD}_{all}$ of *AEDF-Co* also remains close to 1 for different update workloads. This indicates that *AEDF-Co* is not only just able to maintain the quality of a data object above the level as defined by V_i^{max} but it also can achieve a QoD that is close to the maximum quality. The good performance of *AEDF-Co* can also be observed in Fig. 13 in which the QoS of *AEDF-Co* remains close to 1 for different number of update transactions. One of main reasons for the better performance of *AEDF-Co* as compared with *UF* and CF is the lower update workload due to the use of the deferrable scheduling and the look-ahead policy. Although *UF*-min can provide a good

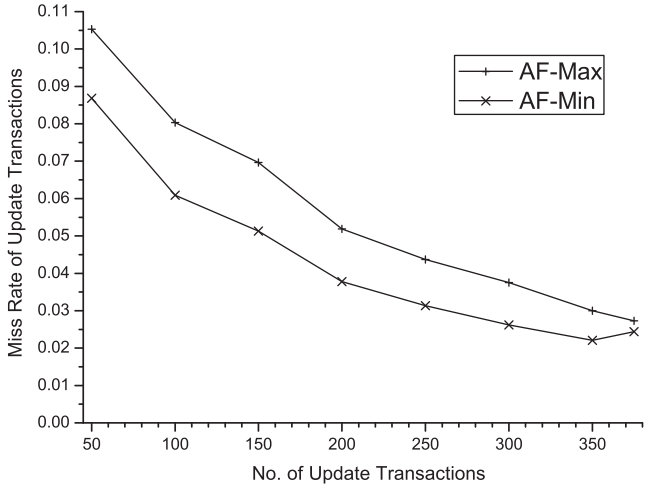

Fig. 11. MR_U vs. # of Update Trans.

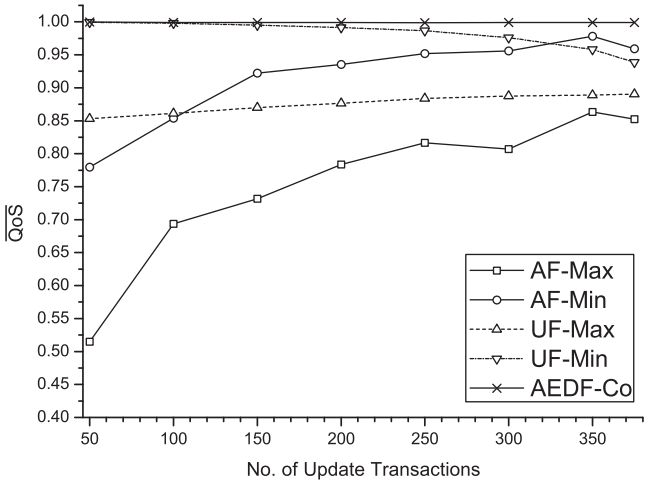

Fig. 13. $\overline{QoS}$ vs. # of Update Trans.

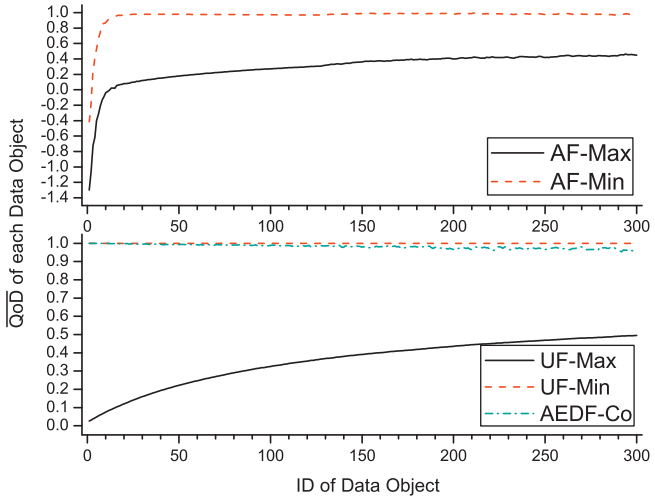

Fig. 14. $\overline{QoD}$ for each Data Object.

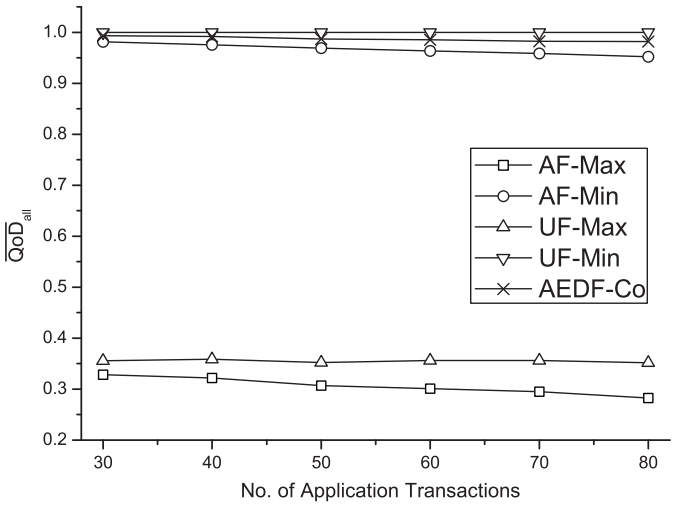

Fig. 16. $\overline{QoD}_{all}$ vs. # of App. Trans.

QoD of data objects, the QoS of *UF*-min is still not as good as *AEDF-Co* as shown in Fig. 13. This is because some of the application jobs miss their deadlines in *UF*-min. Note that once an application job misses its deadline, the QoS it returned will be reduced to zero.

As an example to show the QoD for each data object, we present the results for the experiments in which the number of update transactions is 300 and the number of application transactions is 60. Fig. 14 shows the average QoD of each data object ($\overline{QoD}$). Consistent with the results shown in Figs. 10 and 12, as shown in Fig. 14, in general, the $\overline{QoD}$ of *AF* is worse compared with *UF* and *AEDF-Co* especially for the data objects with smaller IDs. It is because the update transactions which are responsible to maintain the validity of the data objects with smaller IDs are assigned higher priorities in calculating their periods and relative deadlines. Since the calculation uses the worst-case response time of an update transaction to determine its relative deadline, the relative deadlines for higher priority update jobs are tighter. Thus, they are easier to miss deadlines after preemption from application jobs. Not just $\overline{QoD}$, as shown in Fig. 15, the worst-case QoD of a data object ($\overline{QoD}_w$) of *AF* is also much lower compared with *UF* and *AEDF-Co*. It reaches a value of less than −14 for the data objects with smaller IDs. As shown in Figs. 14 and 15, the $\overline{QoD}$ and $\overline{QoD}_w$ of *UF* depend on its chosen values for validity intervals. *UF*-min gives higher values of $\overline{QoD}$ and $\overline{QoD}_w$ as the update transactions have shorter periods for generating update jobs compared with *UF*-max. Consistent with the results presented in Fig. 12, the $\overline{QoD}$ and $\overline{QoD}_w$ of *AEDF-Co* are similar to that of *UF*-min and have values close to 1 for different data objects as shown in Figs. 14 and 15.

6.2.2. Impact of application transaction workload

In the second set of experiments, we vary the number of application transactions while the number of update transactions is fixed at 300. Increasing the number of application transactions increases the total workload of the application transactions. Consistent with the results obtained from the first set of experiments, although *UF* in general gives a better QoD compared with *AF* (Figs. 16–18), the MR_A of *UF* is the worst compared with *AF* and *AEDF-Co* (Fig. 19). On the other hand, although in *AF* the deadlines of all the application jobs can be met as shown in Fig. 19, the QoD observed by the application jobs can be poor as shown in Figs. 18 and 20. The lower $\overline{QoD}_{all}$ of *AF* as shown in Fig. 16 is due to higher miss rate of update jobs in *AF* (MR_U) (Fig. 21). The MR_U of *AF* increases with an increase in application workload as higher application workload increases the preemption from application jobs on the update jobs. Thus, the update jobs have higher probability to miss deadlines in *AF*. Consistent with the results obtained from the first set of experiments, *AEDF-Co* gives the best overall performance in terms of meeting the

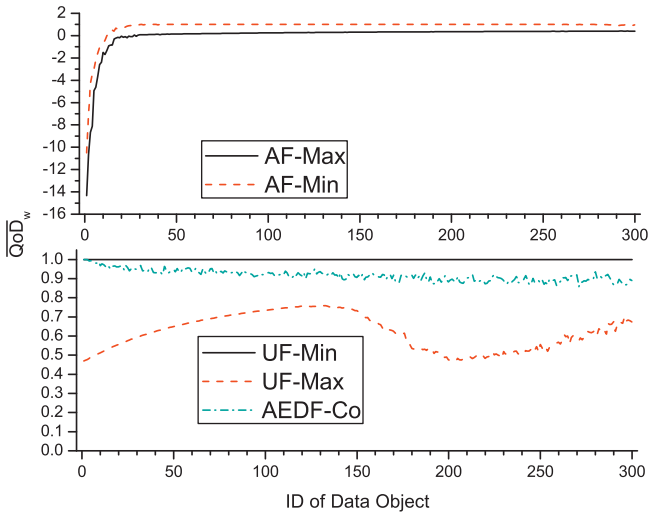

Fig. 15. Worst-Case $\overline{QoD}$.

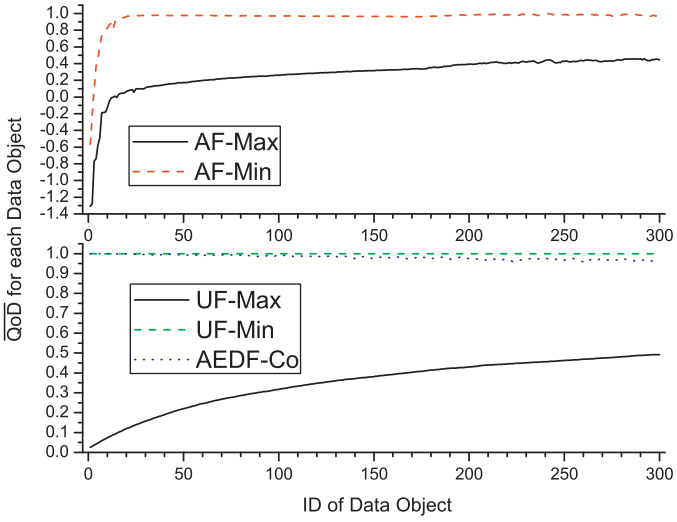

Fig. 17. $\overline{QoD}$ vs. Data Object ID.

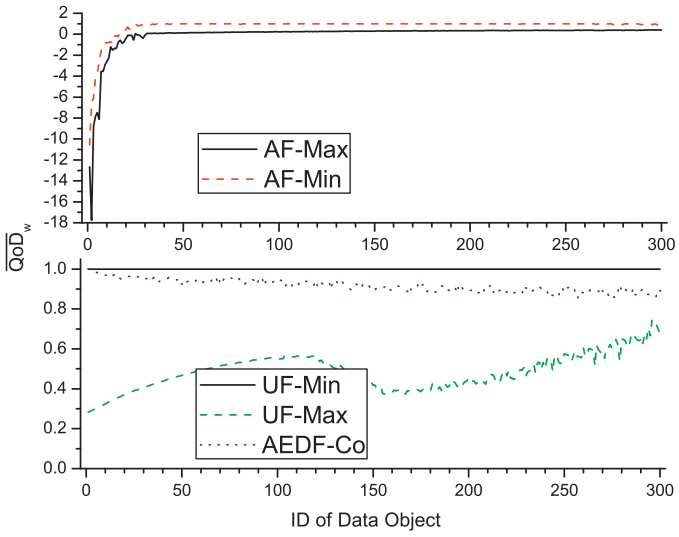

Fig. 18. Worst Case $\overline{QoD}$ vs. Data Object ID.

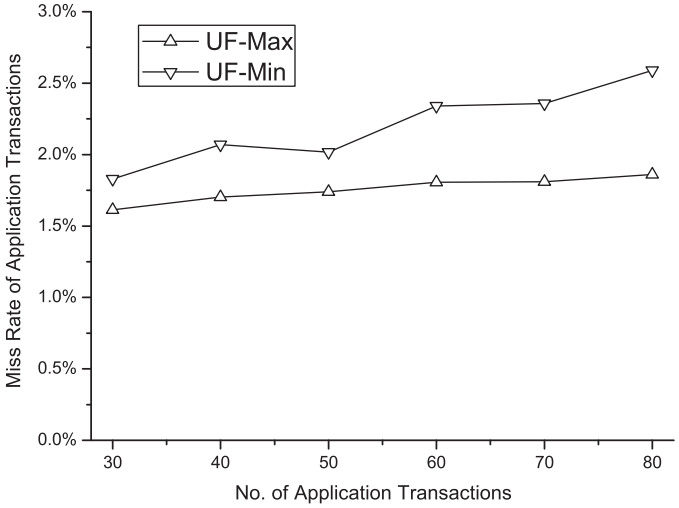

Fig. 19. MR_A vs. # of App. Trans.

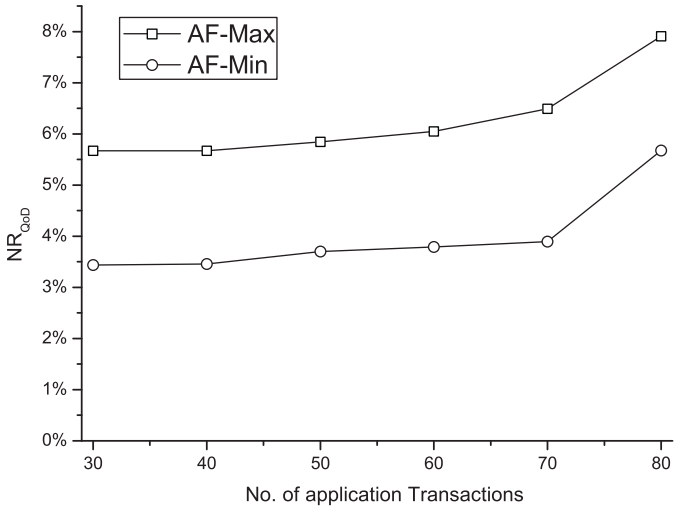

Fig. 20. NR_{QoD} vs. # of App. Trans.

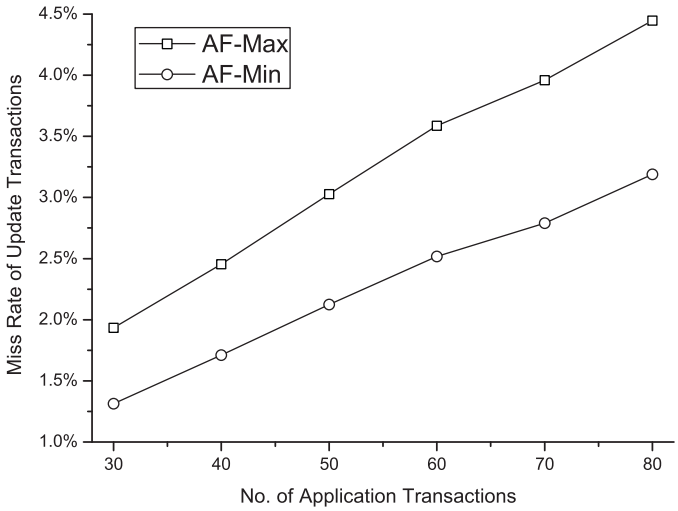

Fig. 21. MR_U vs. # of App. Trans.

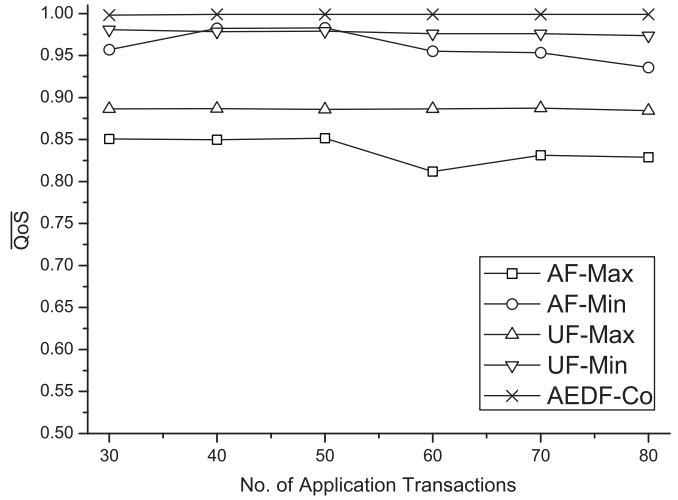

Fig. 22. $\overline{QoS}$ vs. # of App. Trans.

deadlines of the application jobs and maximizing the QoD for execution of the application jobs as shown in Figs. 16–19. The QoS of *AEDF-Co* is also the best for different number of application transactions as shown in Fig. 22.

6.2.3. Interleaved-U/A

Figs. 23–25 show the performance of Interleaved-U/A (both U/A-min and U/A-max) for different values of the relative priority factor as compared with *AEDF-Co*. In U/A-min, the validity interval for a data object is set to be the minimum validity of the data object while in U/A-max, the maximum validities of the data objects are used. The number of update transactions and application transactions are set to be 300 and 60 respectively. As shown in Figs. 23 and 24, increasing the value for the relative priority factor decreases the QoD and QoS as the number of update transactions with priority higher than the application transactions is decreased. At the same time, the miss rate of the application transactions is decreased as shown in Fig. 25. Similarly, U/A-min gives better QoD as compared with U/A-max as the generation periods of the update transactions are smaller. The tradeoff is higher miss rate of application transactions. As shown in Figs. 23–25, *AEDF-Co* consistently gives better performance in term of QoD, QoS and meeting the deadlines of application transactions compared with U/A-min and U/A-max for different values of relative priority factors.

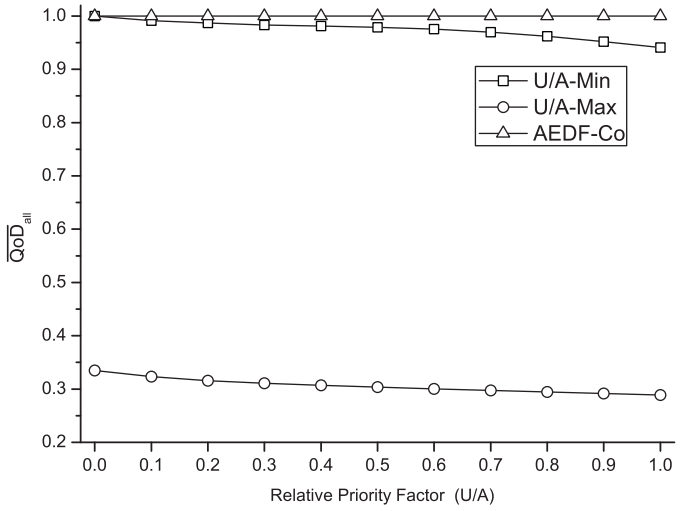

Fig. 23. $\overline{QoD}_{all}$ vs. Relative Priority Factor.

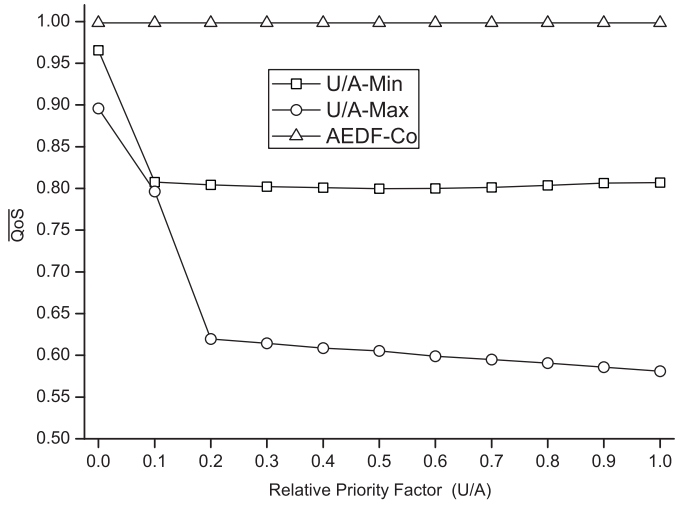

Fig. 24. $\overline{QoS}$ vs. Relative Priority Factor.

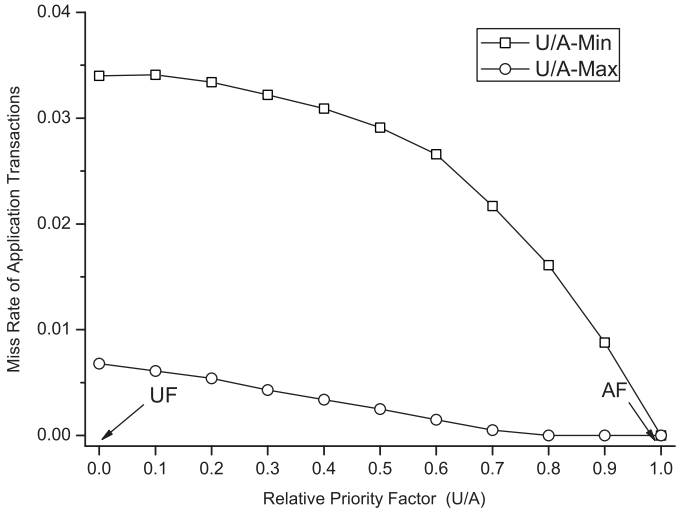

Fig. 25. MR_A vs. Relative Priority Factor.

7. Conclusions and future work

In this paper, we address the co-scheduling problem of update and application transactions in a real-time database system for surveillance of critical events. To ensure the effectiveness of the surveillance functions, we need to meet the deadlines of application jobs and at the same time to provide good quality of data for their executions. To achieve these goals at the same time is difficult. In this paper, we propose a dynamic co-scheduling approach called *Adaptive Earliest Deadline First Co-Scheduling* (*AEDF-Co*) to schedule the update jobs and application jobs. As shown in our experimental results, *AEDF-Co* is effective in meeting the deadlines of all the application jobs and maintain the QoD of the real-time objects to a level close to maximum by timely completion of the update jobs. It is also shown that *AEDF-Co* gives a much better overall performance in meeting the two performance goals as compared with the baseline methods *UF* and *AF* as well as a combined version of them. An important future work is how to apply this co-scheduling technique in wireless sensor and actuator networks where transmission delays between sensors/actuators and the controller cannot be ignored.

References

Abbott, R.K., Garcia-Molina, H., 1992. Scheduling real-time transactions: a performance evaluation. ACM Transactions on Database Systems 17, 513–560.

Adelberg, B., Garcia-Molina, H., Kao, B., 1995. Applying update streams in a soft real-time database system. ACM SIGMOD Record 24, 245–256.

Ahmed, Q., Vrbsky, S., 2000. Triggered updates for temporal consistency in real-time databases. Real-Time Systems 19, 209–243.

Amirijoo, M., et al., 2006. Specification and management of QoS in real-time databases supporting imprecise computations. IEEE Transactions on Computers 55, 304–319.

Bateni, M., Golab, L., Hajiaghayi, M., Karloff, H., 2009. Scheduling to minimize staleness and stretch in real-time data warehouses. In: Proceedings of the Symposium on Parallelism in Algorithms and Architectures, pp. 29–38.

Dertouzos, M.L., 1974. Control robotics: the procedural control of physical processes. In: Proceedings of International Federation for Information Processing Congress, pp. 807–813.

Golab, L., Johnson, T., Shkapenyuk, V., 2009. Scheduling updates in a real-time stream warehouse. In: Proceedings of the International Conference on Data Engineering, pp. 1207–1210.

Gustafsson, T., Hansson, J., 2004. Data management in real-time systems: a case of on-demand updates in vehicle control systems. In: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 182–191.

Han, S., Chen, D., Xiong, M., Mok, A., 2008. A schedulability analysis of deferrable scheduling using patterns. In: Proceedings of the Euromicro Conference on Real-Time Systems, pp. 47–56.

Han, S., Chen, D., Xiong, M., Mok, A.K., 2009. Online scheduling switch for maintaining data freshness in flexible real-time systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 115–124.

HART, F., 2007. Wirelesshart common practice command specification. HART Communication Foundation Document, Number: HCF SPEC-151.

Ho, S.J., Kuo, T.W., Mok, A.K., 1997. Similarity-based load adjustment for real-time data-intensive applications. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 144–153.

Kang, K.D., Oh, J., Son, S.H., 2007. Chronos: feedback control of a real database system performance. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 267–276.

Kang, K.D., Son, S.H., Stankovic, J.A., 2004. Managing deadline miss ratio and sensor data freshness in real-time databases. IEEE Transactions on Knowledge and Data Engineering 16, 1200–1216.

Kang, W., Son, S.H., Stankovic, J.A., 2009. QeDB: a quality-aware embedded real-time database. In: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 108–117.

Ko, J., Lu, C., Srivastava, M., Stankovic, J., Terzis, A., Welsh, M., 2010. Wireless sensor networks for healthcare. Proceedings of the IEEE 98, 1947–2196.

Kulkarni, D., Ravishankar, C., Cherniack, M., 2008. Real-time, load-adaptive processing of continuous queries over data streams. In: Proceedings of the International Conference on Distributed Event-based Systems, pp. 1–4.

Kuo, T.W., Mok, A.K., 1993. Ssp: a semantics-based protocol for real-time data access. In: Proceedings of the IEEE Real-Time Systems Symposium.

Kuo, T.W., Mok, A.K., 1994. Using data similarity to achieve synchronization for free. In: 11th IEEE Workshop on Real-Time Operating Systems and Software, pp. 112–116.

Labrinidis, A., Roussopoulos, N., 2001. Update propagation strategies for improving the quality of data on the web. In: Proceedings of the International Conference on Very Large DataBases, pp. 391–400.

Lam, K.Y., Kuo, T.W., 2001. Real-Time Database Systems: Architecture and Techniques. Kluwer Academic Publishers.
Lam, K.Y., Kuo, T.W., Lee, T.S.H., 2002. Strategies for resolving inter-class data conflicts in mixed real-time database systems. Journal of Systems and Software 61, 1–14.
Lam, K.Y., Xiong, M., Liang, B., Guo, Y., 2004. Statistical quality of service guarantee for temporal consistency of real-time data objects. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 276–285.
Li, M., Liu, Y., 2009. Underground coal mine monitoring with wireless sensor networks. ACM Transactions on Sensor Networks 5, 10:1–10:29.
Liu, C., Layland, J., 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM 20, 46–61.
Nixon, M., Chen, D., Blevins, T., Mok, A., 2008. Meeting control performance over a wireless mesh network. In: Proceedings of the IEEE International Conference on Automation Science and Engineering, pp. 540–547.
Qu, H., Labrinidis, A., 2007. Preference-aware query and update scheduling in web-databases. In: Proceedings of the International Conference on Data Engineering, pp. 356–365.
Ramamritham, K., 1993. Real-time databases. Distributed and Parallel Databases 1, 199–226.
Ramamritham, K., Son, S.H., Dipippo, L.C., 2004. Real-time databases and data services. Real-Time Systems 28, 179–215.
Seto, D., Lehoczky, J.P., Sha, L., Shin, K.G., 1996. On task schedulability in real-time control systems. In: Proceedings of the 17th IEEE Real-Time Systems Symposium, pp. 13–21.
Sha, L., Rajkumar, R., Lehoczky, J.P., 1990. Priority inheritance protocols: an approach to real-time synchronization. IEEE Transactions on Computers 39, 1175–1185.
Shanker, U., Misra, M., Sarje, A.K., 2008. Distributed real time database systems: background and literature review. Distributed and Parallel Databases 23, 127–149.
Song, J., Han, S., Mok, A.K., Chen, D., Lucas, M., Nixon, M., Pratt, W., 2008. WirelessHART: applying wireless technology in real-time industrial process control. In: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 377–386.
Thiele, M., Bader, A., Lehner, W., 2009. Multi-objective scheduling for real-time data warehouses. Computer Science-Research and Development 24, 137–151.
Wei, Y., Prasad, V., Son, S., Stankovic, J.A., 2006. Prediction-based QoS management for real-time data streams. In: Proceedings of the IEEE International Real-Time Systems Symposium, pp. 344–358.
Xiang, J., Li, G.h., Xu, H.J., Du, X.K., 2008. Data freshness guarantee and scheduling of update transactions in RTMDBS. In: Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1–4.
Xiong, M., Han, S., Chen, D., 2006a. Deferrable scheduling for temporal consistency: schedulability analysis and overhead reduction. In: Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 117–124.
Xiong, M., Han, S., Chen, D., Lam, K.Y., Feng, S., 2010. Desh: overhead reduction algorithms for deferrable scheduling. Real-Time Systems 44, 1–25.
Xiong, M., Han, S., Lam, K.Y., 2005. A deferrable scheduling algorithm for real-time transactions maintaining data freshness. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 27–37.
Xiong, M., Han, S., Lam, K.Y., Chen, D., 2008a. Deferrable scheduling for maintaining real-time data freshness: algorithms, analysis, and results. IEEE Transactions on Computers 57, 952–964.
Xiong, M., Liang, B., Lam, K.Y., Guo, Y., 2006b. Quality of service guarantee for temporal consistency of real-time transactions. IEEE transactions on knowledge and data engineering 18, 1097–1110.
Xiong, M., Ramamritham, K., 2004. Deriving deadlines and periods for real-time update transactions. IEEE Transactions on Computers 53, 567–583.
Xiong, M., Wang, Q., Ramamritham, K., 2008b. On earliest deadline first scheduling for temporal consistency maintenance. Real-Time Systems 40, 208–237.

Song Han received the BS degree in computer science from Nanjing University, Peoples Republic of China in 2003 and the MPhil degree in computer science from City University of Hong Kong in 2006. He is currently a PhD candidate in the Department of Computer Science at the University of Texas at Austin. His research interests include cyber-physical systems, real-time and embedded systems, database systems and wireless networks. He is a student member of the IEEE..

Kam-yiu Lam received the BSc (Hons; with distinction) degree in computer studies and the PhD degree from City University of Hong Kong in 1990 and 1994, respectively. He is currently an associate professor in the Department of Computer Science at City University of Hong Kong. His research interests include real-time database systems, real-time active database systems, mobile computing, and distributed multimedia systems.

Jiantao Wang received the BS degree in computer science from University of Science and Technology of China (USTC), People's Republic of China, in 2009. He is currently working toward the PhD degree in the Department of Computer Sciences at City University of Hong Kong. His research interests include real-time systems, real-time data management, and flash-based database systems. He is a student member of the IEEE.

Sang Hyuk Son is a Professor at the Department of Computer Science of University of Virginia. He received the BS degree in electronics engineering from Seoul National University, MS degree from KAIST, and the PhD in computer science from University of Maryland, College Par. He has been a Visiting Professor at KAIST and Sogang University in Korea, City University of Hong Kong, Ecole Centrale de Lille in France, and Linkoping University and University of Skovde in Sweden.

Professor Son is on the executive board of the IEEE Technical Committee on Real-Time Systems, for which he served as the Chair during 2007–2008. He is serving as an Associate Editor for IEEE Transactions on Computers, Real-Time Systems Journal, and Journal of Computing Science and Engineering. He has also served as an Associate Editor of IEEE Transactions on Parallel and Distributed Systems. He is currently on the steering committee of several conferences including Cyber Physical Systems Week, and has served as the Program Chair and General Chair of a number of real-time systems and sensor network conferences, including IEEE Real-Time Systems Symposium. His research interests include real-time systems, database and data services, QoS management, wireless sensor networks, and information security. He has written or co-authored over 280 papers and edited/authored four books in these areas.

Aloysius K. Mok received the BS degree in Electrical Engineering, the MS degree in Electrical Engineering and Computer science, and the PhD degree in Computer Science, all from the Massachusetts Institute of Technology. He is the Quincy Lee Centennial Professor in Computer Science at the University of Texas at Austin, where he has been a member of the faculty of the Department of Computer Sciences since 1983. He has performed extensive research on computer software systems and is internationally known for his work in real-time systems. He is a past chairman of the Technical Committee on Real-Time Systems of the IEEE and has served on numerous national and international research and advisory panels. His current interests include real-time and embedded systems, robust and secure network centric computing, and real-time knowledge-based systems. In 2002, Dr. Mok received the IEEE Technical Committee on Real-Time Systems Award for his outstanding technical contributions and leadership achievements in real-time systems. He is a member of the IEEE.