Multiprocessor Scheduling of Sensor Transactions for Real-Time Data Quality Maintenance

Tian Bai, Zhijie Li, and Bo Fan

School of Information Science and Engineering, Hunan Institute of Science and Technology, Yueyang 414000, China

Correspondence should be addressed to Bo Fan; bondfan@163.com

Received 19 April 2020; Revised 14 August 2020; Accepted 31 August 2020; Published 21 September 2020

Academic Editor: Paolo Bellavista

In cyber-physical systems, sensor transactions should be effectively scheduled to maintain the temporal validity of real-time data objects. Previous studies on sensor transaction scheduling mainly focus on uniprocessor systems. In this paper, we study the problem of data quality-based scheduling of sensor transactions on multiprocessor platforms. The data quality is defined to describe the validity degree of real-time data objects. Two methods, named the Partitioned Scheduling for Quality Maximization (P-QM) and the improved P-QM scheduling (IP-QM), are proposed. P-QM maximizes the data quality by judiciously determining the preallocated computation time of each sensor transaction and assigns the transactions to different processors. IP-QM improves the data quality obtained from P-QM by adaptively executing transaction instances on each processor based on the current status of the system. It is demonstrated through experiments that IP-QM can provide higher data quality than P-QM under different system workloads.

1. Introduction

Cyber-physical systems (CPS) feature a tight combination of the computational and physical elements of the systems [1–3]. They are widely used in applications that need to process real-time data in a timely manner. Example applications include road traffic control and industry process control [4–6]. The real-time data objects model the current status of entities in a system environment. Different from the traditional data objects, their values may become invalid with the passage of time. Associated with each real-time data object is a validity interval that specifies the lifetime of its current value. If this lifetime does not expire, the data object is temporally valid. Otherwise, it becomes invalid and a new data value needs to be installed.

In CPS, sensor transactions are generated to sample the status of external entities and update the corresponding data values. They should be effectively scheduled to maintain the temporal validity of real-time data objects. This scheduling problem consists of two issues. That is, the determination of the release time and the deadline of each transaction instance and the scheduling of these instances. Various methods have been proposed to solve this problem. Some examples are the More-Less scheme (ML), the deferrable scheduling algorithm with fixed priority (DS-FP), and HS$_{EDF}$[7–9]. These methods aim to reduce the update workload while providing a complete guarantee on temporal validity, i.e., guaranteeing that the data objects are valid all the time. The update workload reduction allows a system to consume less energy. In addition, it leaves more processor resources to other types of transactions, such as user transactions and triggered transactions.

For many systems, providing completely guaranteed temporal validity for real-time data objects could be difficult. At first, the user transactions in the system may have strict timeliness requirements. They compete with the sensor transactions for the same set of resources for execution. To guarantee temporal validity, more resources should be given to sensor transactions. This could lead to high deadline miss ratio of user transactions. Secondly, the system workload can be highly dynamic. The computation time of a transaction in the worst case can be much larger than that in the normal case. The arrivals of transaction instances may be aperiodical and unpredictable. As a result, data validity violations may
occurrences during system overloads. To tackle these problems, several quality of service- (QoS-) based methods have been proposed to schedule sensor transactions and user transactions to maintain the quality of real-time data objects and the quality of transactions [10–22].

Current methods for sensor transaction scheduling are mainly restricted to uniprocessor systems. In this paper, we study the problem of data quality-based scheduling of sensor transactions on multiprocessor platforms. We consider the partitioned scheduling approach. On each processor, the earliest deadline first (EDF) scheme is adopted. The major contributions of the paper are as follows:

(i) A definition of data quality is presented to describe the validity degree of real-time data objects. The definition considers the validity of individual data objects and the validity of correlated data object sets.

(ii) Two scheduling methods, named the Partitioned Scheduling for Quality Maximization (P-QM) and the improved P-QM scheduling (IP-QM), are proposed to maximize the data quality.

(iii) Experiments are conducted to evaluate the performance of the proposed methods. The results show that IP-QM outperforms P-QM in terms of the average quality of individual data objects and the average quality of correlated sets.

The rest of the paper is organized as follows. Section 2 reviews previous studies on temporal validity maintenance. Section 3 describes the system model. The definition of data quality is also presented in this section. The details of the P-QM method and the IP-QM method are presented in Section 4 and Section 5, respectively. Performance studies are given in Section 6. Finally, Section 7 concludes the paper.

2. Related Studies

In recent years, there have been a number of studies on maintaining temporal validity of real-time data. ML adopts the periodic task model and the deadline monotonic scheme (DM) [7]. In ML, a sensor transaction’s period is set to be no shorter than its relative deadline. The sum of the period and the deadline is equal to the validity interval. The Half-Half scheme (HH) can be viewed as a special case of ML [23]. DS-FP adopts a sporadic task model [8]. It reduces the processor workload by judiciously deferring the release times of transaction instances. Two extensions of DS-FP were presented in [24] to reduce the online scheduling overhead. The basic idea is to produce a hyperperiod of the DS-FP schedule and repeat the hyperperiod infinitely. A necessary and sufficient schedulability condition for DS-FP in discrete time systems was proposed in [25]. The problem of temporal validity maintenance under dynamic priority scheduling was first studied in [9]. Three algorithms were proposed to derive the periods and deadlines of sensor transactions under the earliest deadline first scheme (EDF). A two-phase algorithm was proposed in [26] to reduce the searching cost of period and deadline assignment under EDF. Li et al. [27] presented two methods to maintain temporal validity for EDF when transmission delays are considered. DS-FP was extended in [25] to be a dynamic priority scheduling algorithm by applying EDF to schedule update instances. The problem of scheduling tasks with both maximum distance constraints (i.e., temporal validity constraints) and minimum distance constraints was investigated in [28]. Jha et al. [29] investigated how to maintain the mutual temporal consistency of real-time data objects. Han et al. studied the problem of maintaining temporal validity in the presence of mode changes [30]. Two algorithms were presented to search for proper mode switch points. These studies are all limited to uniprocessor systems.

In [31], Li et al. proposed several algorithms to partition a set of sensor transactions on multiprocessors under EDF and DM. The resource augmentation bounds of these algorithms were also derived. The global EDF algorithm and the half-half principle were applied in [32] to satisfy the data validity constraints. The energy-aware real-time data processing problem on multicore platforms was studied in [33]. Efficient techniques were proposed to maintain temporal validity while reducing energy cost.

The studies described above focus on providing a complete guarantee on temporal validity. The co-scheduling problem of periodic application transactions and update transactions was investigated in [10–12]. The aim is to meet the deadlines of all the application transactions while maximizing the quality of data objects. The algorithms based on fixed priority scheduling scheme, EDF and DS-LALF, were presented, respectively. In [13], a set of extensions of ML was proposed to achieve the trade-off between QoS of temporal validity and the number of supported transactions. Labrinidis and Roussopoulos studied the problem of maintaining the freshness of views on a web server [14]. A quality-aware update scheduling algorithm was proposed based on the popularity of views. In [15], a QoS management architecture was proposed to support the desired QoS by applying feedback control, admission control, and flexible freshness management. On-demand schemes were proposed in [16] to skip the updates with similar data object values. Amirjoo et al. employed the notion of imprecise computation for QoS specification and management [17]. Differentiated data service approaches for transaction classes with diverse importance and QoS requirements were proposed in [18]. An effective approach was presented in [19] to decrease both the deadline miss ratio and power consumption by real-time query aggregation and data freshness adaptation. All the above QoS-based methods are designed for uniprocessor systems.

In [20], a scheduling framework was presented to assign update jobs to multiple tracks and schedule them on each track, with the objective of minimizing the total staleness of data objects in a streaming warehouse. A track here represents a fraction of the computing resources. Bateni et al. proposed an update scheduling algorithm upon multiprocessors that has bounded stretch and weighted staleness under the quasiperiodic model [21]. In [22], Kang and Chung proposed a multiple inputs/multiple outputs (MIMO) feedback control method to support the timeliness of data-intensive tasks running on multicore-based
3. System Model

Let $X = \{X_i\}_{i=1}^n$ denote a set of real-time data objects and $\Gamma = \{\tau_i\}_{i=1}^m$ a set of sensor transactions. The data object $X_i (1 \leq i \leq n)$ is associated with a validity interval $V_i$. It is value is updated by transaction $\tau_i (1 \leq i \leq n)$. $\tau_i$ consists of a sequence of instances (update jobs), in which each samples a value of $X_i$ and installs it. The $j$th instance of $\tau_i$ is denoted as $\tau_{i,j}$. We assume that the jitter between the sampling time and the release time of an instance is zero. $g_i(t)$ is the probability density function that describes the distribution of the actual computation times of $\tau_i$’s instances. Transactions in $\Gamma$ are indexed according to nondecreasing order of their validity intervals, i.e., $V_i \leq V_{i+1}$ for all $i$, $1 \leq i < n$.

The transaction set $\Gamma$ is scheduled upon a multiprocessor platform $\Pi = \{\pi_i\}_{i=1}^m$. In general, there are three approaches to schedule $\Gamma$: partitioned approach, global approach, and hybrid approach [34]. In the partitioned approach, each transaction is assigned to a processor and is always executed on it. In the global approach, an instance that has been preempted on one processor can resume its execution on a different processor. The hybrid approach is the combination of partitioned and global approach. It can be further classified into semi-partitioned approach and cluster-based approach. In this paper, the partitioned approach adopted. On each processor, the EDF scheme is used to schedule the transaction instances.

Definition 1. $X_i$ is temporally valid at time $t$ if, for its update job finished latest before $t$, the sampling time of this job $t_r$ plus $V_i$ is not less than $t$, i.e., $t_r + V_i \geq t$ [7].

According to Definition 1, if $X_i$’s current value is sampled at $t_r$, it should be updated before $t_r + V_i$. Otherwise, it will become invalid. However, when the actual computation time of the corresponding update job is large, the update may not be finished before $t_r + V_i$ due to the lack of processor resources. Consider a time period $[0, T]$. Let $v_i(t)$ denote the valid state of $X_i$ at time $t (0 \leq t \leq T)$. $v_i(t)$ is 1 if $X_i$ is valid at $t$ and is 0 otherwise. The quality of $X_i$, $q(X_i)$, is defined as follows:

$$q(X_i) = \frac{1}{T} \int_0^T v_i(t) \, dt.$$  (1)

A correlated data object set is a set of real-time data objects whose values are used together to compute the corresponding derived data or to make decisions. It is often required that a certain percentage of the data objects in the correlate set are valid to produce the result with sufficient accuracy. Let $Y = \{Y_k \mid Y_k \subseteq X\}_{k=1}^n$ denote the correlated set in the system. For each $Y_k (1 \leq i \leq n)$, let $N_{th}(Y_k)$ denote the valid threshold of $Y_k$. If the number of data objects in $Y_k$ that are valid at time $t$ is no less than $N_{th}(Y_k)$, then $Y_k$ is considered to be valid at $t$. $N_{th}(Y_k)$ is set by users based on application requirements. The quality of $Y_k$, $q(Y_k)$, is defined as follows:

$$q(Y_k) = \frac{1}{T} \int_0^T \left( \sum_{X_i \in Y_k} v_i(t) \geq N_{th}(Y_k) \right) \, dt.$$  (2)

The overall data quality of the system, $q(X)$, is defined as follows:

$$q(X) = \sum_{X_i \in X} q(X_i) + \sum_{Y_k \in Y} q(Y_k).$$  (3)

Sensor transactions should be effectively scheduled on $\Pi$ to maximize $q(X)$. This scheduling problem consists of two subproblems. The first is to assign the sensor transaction to processors in a data quality-aware manner. Notice that, different from the traditional real-time tasks, the periods and deadlines of sensor transaction are unknown. They must also be derived during the assignment. The second is to determine which instances can be executed in the system and how to execute them. Table 1 summarizes the major symbols that are used in the paper.

4. The P-QM Method

In this section, we present the P-QM method. Instead of maximizing $q(X)$ directly, P-QM tries to maximize an approximated overall data quality. Let $C^p = \{C^p_i\}_{i=1}^3$ denote the preallocated computation times of transactions in $\Gamma$. The actual computation time of instance $\tau_{i,j}$ is denoted as $c_{i,j}$. The quality of $X_i$ with respect to $C^p$, $q(X_i \mid C^p)$, is defined as the probability that $c_{i,j}$ is no larger than $C^p$, i.e., $Pr(c_{i,j} \leq C^p)$. For a correlated set $Y_k$, let $C^p(k)$ denote the preallocated computation times of the corresponding transactions. The quality of $Y_k$ with respect to $C^p(k)$, $q(Y_k \mid C^p(k))$, is defined as follows:

$$q(Y_k \mid C^p(k)) = \sum_{S \subseteq Y_k} \prod_{X_i \in S} Pr(c_{i,j} \leq C^p) \prod_{X_i \in Y_k \setminus S} \left(1 - Pr(c_{i,j} \leq C^p)\right).$$  (4)

In equation (4), $S = \{S \subseteq Y_k \mid |S| \geq N_{th}(Y_k)\}$. $q(Y_k \mid C^p(k))$ can be viewed as the probability that the number of data objects in $Y_k$ that are valid at a time instant is no less than $N_{th}(Y_k)$ under $C^p(k)$. The overall data quality of the system with respect to $C^p$, $q(X \mid C^p)$, is defined as follows:

$$q(X \mid C^p) = \sum_{X_i \in X} q(X_i \mid C^p) + \sum_{Y_k \in Y} q(Y_k \mid C^p(k)).$$  (5)

P-QM maximizes $q(X \mid C^p)$ by judiciously determining $C^p$ and assigning the transactions to processors. A transaction assignment method under the given $C^p$ is presented at
Table 1: Definition of symbols.

| Symbol | Definition |
|--------|------------|
| $X_i$  | The $i$th real-time data object |
| $V_i$  | The validity interval of $X_i$ |
| $\pi_i$ | The $i$th processor |
| $\tau_i$ | The sensor transaction used to update $X_i$ |
| $\tau_{ij}$ | The $j$th instance of $\tau_i$ |
| $P_i$  | The period of $\tau_i$ |
| $D_i$  | The deadline of $\tau_i$ |
| $C_i^p$ | The preallocated computation time of $\tau_i$ |
| $Y_k$  | The $k$th correlated data object set |
| $q(X_i)$ | The quality of $X_i$ |
| $q(Y_k)$ | The quality of $Y_k$ |
| $q(X)$  | The overall data quality of the system |
| $\lambda_i$ | The density of $\tau_i$ |
| $\Delta_i$ | The cumulative density of $\tau_i$ |
| $\lambda_{\text{max}}$ | The maximum density among $\lambda_1, \lambda_2, \ldots, \lambda_n$ |

4.1. Assigning Transactions to Processors. In P-QM, sensor transactions are assigned to processors in a way that the temporal validity of data objects in $X$ is completely guaranteed under the given $C_i^p$. Let $P_i$ and $D_i$ denote the period and the deadline of transaction $\tau_i$, respectively. The assignment problem is described as follows.

Given the transaction set $\Gamma$ with $C_i^p$ and the multiprocessor platform $\Pi$, assign each $\tau_i$ to a processor in $\Pi$ and derive $P_i$ and $D_i$, such that the following constraints are satisfied:

1. **Validity constraints**: $\forall i, 1 \leq i \leq n, P_i + D_i \leq V_i$

2. **Deadline constraints**: $\forall i, 1 \leq i \leq n, C_i^p \leq \min\{P_i, D_i\}$

3. **Feasibility constraints**: $\forall j, 1 \leq j \leq m$, the transactions assigned to $\pi_j$ with given preallocated computation times and derived deadlines and periods are feasible by using EDF scheduling.

Notice that if the above constraints are all satisfied, then the temporal validity of data objects is guaranteed. This is because for each $X_i$, its $j$th ($j \geq 2$) value sampled at $(j - 1)P_i$ is updated by instance $\tau_{ij}$ which will finish before $(j - 2)P_i + V_i$. This means a value is certain to be refreshed before its validity interval expires.

Next, we present the algorithm used in P-QM to solve the assignment problem. Let $\Delta_i$ and $\lambda_i$ denote the cumulative density and the density of $\tau_i$ with respect to $C_i^p$, respectively. $\Delta_i = \sum_{j \in \mathcal{C}} (V_j - 2C_i^p)/C_i^p$, $\lambda_i = C_i^p/V_i$. Let $\Delta_{\text{max}} = \max\{\Delta_i | 1 \leq i \leq n\}$, $\lambda_{\text{sum}} = \sum_{1 \leq i \leq n} \lambda_i$, and $\Delta_{\text{max}} = \max\{\Delta_i | 1 \leq i \leq n\}$. The following conditions are checked at first:

$$m \geq \frac{2(\Delta_{\text{max}} + \lambda_{\text{sum}} - \lambda_{\text{max}})}{1 - 2\lambda_{\text{max}}}$$  \hspace{1cm} (6)

$$\Delta_{\text{max}} + \lambda_{\text{sum}} \leq \frac{1}{2}$$  \hspace{1cm} (7)

If equation (6) or equation (7) holds, the assignment mode is set to be the restricted mode. In this mode, transaction $\tau_i$'s deadline is restricted to be no larger than $V_i/2$. Otherwise, it is set to be the unrestricted mode. In this mode, $\tau_i$'s deadline can be larger than $V_i/2$ but should be no larger than $V_i$. Suppose the first $i$ transactions have been successfully assigned to processors. Let $\Gamma_k$ denote the set of transactions that are assigned to processor $\pi_k$. The utilization of $\tau_i$ is $u_i = (C_i^p/P_i)$. If $|\Gamma_k| = 0$, the deadline of $\tau_i$ on $\pi_k$ is set to be $C_i^p$. Otherwise, it is computed as follows:

$$D_i = \max\{D_j | \tau_j \in \Gamma_k\} + \frac{C_i^p}{1 - \sum_{\tau_j \in \Gamma_k} u_j}.$$  \hspace{1cm} (8)

The period of $\tau_i$ is set to be $V_i - D_i$. $\tau_j$ is assigned to the first processor $\pi_k$ such that the following conditions are satisfied:

1. $D_i \leq V_i/2$ if the assignment mode is restricted mode, or $D_i \leq V_i - C_i^p$ if the assignment mode is unrestricted mode.

2. $\sum_{\tau_j \in \Gamma_k} u_j + u_i \leq 1$.

The assignment fails if no such processor exists. This process is described in Algorithm 1. Notice that both the calculation of $\tau_i$'s deadline and the check of condition 2 can be carried out in an incremental way. Thus, the time required to assign $\tau_i$ to a processor is $O(m)$. The check of equations (6) and (7) takes $O(n)$ time. Therefore, the time complexity of Algorithm 1 is $O(mn + n)$. The following theorem shows the correctness of the algorithm.

**Theorem 1.** If Algorithm 1 succeeds, then the temporal validity of data objects in $X$ is completely guaranteed.

**Proof.** For each $\tau_i \in \Gamma$, when it is assigned to a processor, its deadline $D_i$ is set to be no less than $C_i^p$ and no larger than $V_i - C_i^p$ due to equation (8) and condition 1 of the algorithm. Its period $P_i$ is set to be $V_i - D_i$. Thus, the validity constraints and the deadline constraints are satisfied. We only need to show the feasibility constraints are also satisfied.

The first transaction $\tau_i$ is assigned to processor $\pi_i$ with deadline $C_i^p$ and period $V_i - C_i^p$. Obviously, it is EDF-schedulable. Suppose that the first $i - 1$ transactions have been assigned to processors and the transactions on each processor are EDF-schedulable. Consider that transaction $\tau_i$ is assigned to processor $\pi_k$. The transaction set on processors except $\pi_k$ are not affected by $\tau_i$; thus, they are still EDF-schedulable. If $\tau_i$ is the first transaction assigned to $\pi_k$, then obviously it is EDF-schedulable. Let $h^0(\tau_i, t)$ denote the approximated demand bound of $\tau_i$ in $[0, t)$. $h^1(\tau_i, t)$ is $C_i^p + u_i \cdot (t - D_i)$ if $t \geq D_i$ and is 0 otherwise. Notice that $D_i$ is larger than the deadlines of transactions that are assigned to $\pi_k$ prior to $\tau_i$. According to [35], $\Gamma_k \cup \{\tau_i\}$ is EDF-schedulable if $\sum_{\tau_j \in \Gamma_k} u_j + u_i \leq 1$ and

$$D_i - \sum_{\tau_j \in \Gamma_k} h^1(\tau_j, D_j) \geq C_i^p.$$  \hspace{1cm} (9)
\begin{algorithm}
\textbf{Input} : \( \Gamma, C^p, m \)
\textbf{Output} : The assigned processor of each transaction
1. assignment mode \(-\) restricted if equation (6) or (7) holds. Otherwise assignment mode \(-\) unrestricted;
2. for \( i = 1 \) to \( n \) do
3. \hspace{1em} for \( j = 1 \) to \( m \) do
4. \hspace{2em} \( D_i \leftarrow C^p_i \) if \( |\Gamma_j| = 0 \). Otherwise compute \( D_i \) using equation (8);
5. \hspace{2em} \( P_i \leftarrow V_i - D_i \);
6. \hspace{2em} if condition 1 and condition 2 are satisfied then
7. \hspace{3em} \( \Gamma_j \leftarrow \Gamma_j \cup \{ \tau_i \} \);
8. \hspace{2em} break;
9. \hspace{2em} end if
10. \hspace{1em} end for
11. if \( j = m + 1 \) then
12. \hspace{1em} \( \Gamma \) is infeasible on \( \Pi \) under \( C^p \), return failure;
13. \hspace{1em} end if
14. end for
15. return success;
\end{algorithm}

Let \( \Gamma_{k,j} \) denote the transactions in \( \Gamma_k \cup \{ \tau_j \} \) that are assigned to \( \pi_k \) before \( \tau_j \). Based on equation (8), one has
\[
\sum_{r_j \in \pi_k} h^t(r_j, D) = \sum_{r_j \in \pi_k} \left( C^p_j + \sum_{e_i \in \{ e_i \}} \frac{C^p_i u_j}{1 - \sum_{e_i \in \{ e_i \}} u_j} - \sum_{e_i \in \{ e_i \}} \frac{C^p_i u_j}{1 - \sum_{e_i \in \{ e_i \}} u_j} \right) \\
= \sum_{r_j \in \pi_k} \frac{C^p_j}{1 - \sum_{e_i \in \{ e_i \}} u_j} + \sum_{e_i \in \{ e_i \}} \frac{C^p_i u_j}{1 - \sum_{e_i \in \{ e_i \}} u_j} \\
= D_i - C^p_i. \tag{10}
\]

Equation (10) means the transaction set on \( \pi_k \) is EDF-schedulable after \( \tau_j \) is added. By induction, the feasibility constraints are satisfied when all transactions have been assigned to processors.

**Theorem 2.** Algorithm 1 succeeds if equation (6) or equation (7) holds.

**Proof.** At first, we consider the case in which equation (6) holds and equation (7) does not hold. Suppose in this case, transaction \( \tau_j \) fails to be assigned to any processor. Then, on each processor, one or both of the conditions of the algorithm are not satisfied. Let \( \Pi_1 \) denote the set of processors on which condition 1 is not satisfied and \( \Pi_2 \) denote the set of processors on which condition 1 is satisfied and condition 2 is not satisfied. \( |\Pi_1| + |\Pi_2| = m \). The transactions assigned to processors in \( \Pi_1 \) and \( \Pi_2 \) are denoted as \( \Gamma_1 \) and \( \Gamma_2 \), respectively. For each \( \pi_k \in \Pi_1 \), it must be
\[
\max \left\{ D_j \mid \tau_j \in \Gamma_k \right\} + \frac{C^p_j}{1 - \sum_{r_j \in \pi_k} u_j} > \frac{V_i}{2}. \tag{11}
\]

According to Theorem 1, equation (11) implies
\[
C^p_i + \sum_{r_j \in \pi_k} \left( \frac{C^p_j}{P_j} \right) \left( \frac{V_i - D_j}{2} \right) > \frac{V_i}{2}. \tag{12}
\]

Notice that, for each \( \tau_j \in \Gamma_k \), \( V_j / 2 \leq P_j \leq V_j - C^p_j \). Summing over all processors in \( \Pi_1 \) and making some transformations, we obtain
\[
|\Pi_1| \leq \frac{2 \left( \sum_{r_j \in \pi_k} C^p_j / V_i - 1 / V_i \sum_{r_j \in \pi_k} (C^p_j / V_j) (V_j - C^p_j) + \sum_{r_j \in \pi_k} (C^p_j / V_j) \right)}{1 - 2(C^p_j / V_j)}. \tag{13}
\]

For each \( \pi_k \in \Pi_2 \), it must be
\[
\sum_{r_j \in \pi_k} u_j + \frac{2C^p_j}{V_i} > 1. \tag{14}
\]

Summing over all processors in \( \Pi_2 \) and making some transformations, we obtain
\[
|\Pi_2| \leq \frac{2 \left( 2C^p_j / V_j \right)}{1 - 2(C^p_j / V_j)}. \tag{15}
\]

If both \( |\Pi_1| \) and \( |\Pi_2| \) are larger than zero, then based on equations (13) and (15) and the definition of \( \Delta_{\text{max}} \) and \( \lambda_{\text{sum}} \), we obtain
\[
m < \frac{2 \left( \Delta_{\text{max}} + \lambda_{\text{sum}} - \lambda_j \right)}{1 - 2\lambda_j}. \tag{16}
\]

Since equation (7) does not hold, it must be
\[ m < \frac{2(\Delta_{\text{max}} + \lambda_{\text{sum}} - \lambda_{\text{max}})}{1 - 2\lambda_{\text{max}}} \]  

(17)

If \(|\Pi_1| = 0\), we also obtain equation (17). If \(|\Pi_1| = 0\), then \(\lambda_{\text{sum}} > 1/2\), otherwise equation (15) does not hold. We then obtain

\[ m < \frac{2(\lambda_{\text{sum}} - \lambda_{\text{max}})}{1 - 2\lambda_{\text{max}}} \]  

(18)

Both equations (17) and (18) contradict the assumption that equation (6) holds.

Then, we consider the case in which equation (7) holds. Suppose that the first \(i - 1\) transactions have been successfully assigned to processors. For each \(\pi_k \in \Pi\), it must be

\[ C_k^p + \sum_{\tau \in \Pi_k} \left( C_k^p + \frac{C_j^p}{P_j} \left( \frac{1}{2} - D_j \right) \right) \leq (2\Delta_{\text{max}} + 2\lambda_{\text{sum}}) \frac{V_j}{2} \leq \frac{V_j}{2} \]  

(19)

Equation (19) means the deadline of \(\tau_k\) is not larger than \(V_j/2\); thus, condition 1 of the algorithm holds. Condition 2 of the algorithm also holds since equation (7) holds implies \(\lambda_{\text{sum}} \leq (1/2)\). Therefore, each transaction can be assigned to a processor.

Notice that there may exist some transactions in \(\Gamma\) with \(\lambda_i = (1/2)\). Let \(\Gamma_h\) denote the set of such transactions. If \(|\Gamma_h| > m\), or \(|\Gamma_h| = m\) and \(n \geq |\Gamma_h| + 1\), then \(\Gamma\) is infeasible on \(\Pi\) since a processor can only accommodate just one transaction from \(\Gamma_h\). If \(|\Gamma_h| < m\), then Theorem 2 can be applied to \(\Gamma / \Gamma_h\) and \(m - |\Gamma_h|\).

4.2. Determining the Preallocated Computation Times

The problem of determining the preallocated computation times \(C_k^p\) is formulated as an optimization problem:

\[
\max q(X | C_p),
\]

subject to

(1) Feasibility constraint: the transactions in \(\Gamma\) can be successfully assigned to processors under \(C_p\) by Algorithm 1

(2) Computation time constraint: \(\forall i, 1 \leq i \leq n, C_{i_{\text{min}}} \leq C_{i_p} \leq C_{i_{\text{max}}}\)

Theorem 3. Let \(X_i^k\) denote the \(i\)th data object in \(Y_k\), \(C_{i_p}^{k,p}\) the preallocated computation time of \(X_i^k\). Given two sets of preallocated computation times \(C_{i,p}^{k,1}(k)\) and \(C_{i,p}^{k,2}(k)\) of \(Y_k\). If for each \(k\), \(1 \leq k \leq |Y_k|\), \(C_{i,p}^{k,1} \leq C_{i,p}^{k,2}\), then \(q(Y_k | C_{i,p}^{k,1}(k)) \leq q(Y_k | C_{i,p}^{k,2}(k))\).

Proof. Let \(q(i, j, 1)\) and \(q(i, j, 2)\) denote the quality of \(\{X_i^k\}_{i=1}^{|Y_k|}\) with a valid threshold of \(j\) under \(C_{i,p}^{k,1}(k)\) and \(C_{i,p}^{k,2}(k)\), respectively. For each \(i, 1 \leq i \leq |Y_k|\), it must be

(1) For all \(j, 1 \leq j \leq h\), \(q(i, j, 1) < q(i, j, 1)\) and \(q(i, j, 2) < q(i, j, 1, 2)\) and \(q(X_i^k | C_{i_p}^{k,1}) \leq q(X_i^k | C_{i_p}^{k,2})\)

In equation (22), \(q(i, j, 2) - q(i, j, 1) = (q(i, j, 1) - q(i, j, 1, 1)) + (q(i, j, 1, 1) - q(i, j, 1, 2))\). By induction, we know that \(q(|Y_k|, N_{\text{th}}(Y_k), 2) \geq q(|Y_k|, N_{\text{th}}(Y_k), 1)\).

According to Theorem 3, \(q(X | C_p)\) will increase by increasing the preallocated computation times of some transactions. Based on this observation, P-QM adopts a greedy heuristic to determine \(C_p\). For each \(\tau_i\), the heuristic sets the initial value of \(C_{i_p}\) to be its minimum computation time at the start. It then selects one transaction at a time to increase its preallocated computation time.

Let \(\Delta_{q_i}\) denote the increase in the overall data quality due to an increase of current \(C_{i_p}\) by the step size of \(\delta\). \(\Delta_{q_i}\) is computed as follows:

\[
\Delta_{q_i} = q(X_i | C_p + \delta) - q(X_i | C_p) 
+ \sum_{X_i \in Y_k} \left(q(Y_k | C_{i,p}^{k,1}(k))\right)
- q(Y_k | C_p^{k,2}(k))) 
\]

(23)

In every step, transaction \(\tau_i\) is selected to increase \(C_{i_p}\) if \(\Delta_{q_i} = \max \{\Delta_{q_i} | C_{i_p}^{k,1}(k) \cup \{C_{i_p}^{k,2}(k) + \delta\}\}\) is feasible.

To determine whether increasing \(C_{i_p}\) by \(\delta\) is feasible, the following condition is checked at first:

\[
\sum_{1 \leq i \leq |Y_k|, i \neq i} \frac{C_{i_p}^{k,1}}{V_j - C_{i_p}^{k,1} + V_i - C_{i_p}^{k,2}} > m 
\]

(24)

If equation (24) holds, then increasing \(C_{i_p}\) by \(\delta\) is infeasible. To see why, let us suppose increasing \(C_{i_p}\) by \(\delta\) is feasible; then, for each \(\tau_j\), there must exist at least one
instance of \( r_j \) finished in any interval with length \( V_j - C_j^p \), otherwise the data object \( X_j \) will be invalid at some time points. This is also true for \( r_i \) with \( C_i^p + \delta \). The total time required to execute the instances should not exceed the platform capacity. Thus, it must be
\[
m(V_i - C_i^p - \delta) \prod_{1 \leq j \leq n} (V_i - C_i^p) \geq (C_i^p + \delta) \prod_{1 \leq j \leq n} (V_i - C_i^p) + \sum_{1 \leq j < k \leq n} C_i^p (V_i - C_i^p - \delta) \prod_{1 \leq j \leq n \neq k} (V_i - C_i^p).
\] (25)

Dividing both sides of the negation of equation (25) by \( (V_i - C_i^p - \delta) \prod_{1 \leq j < k \leq n} (V_i - C_i^p) \), we obtain equation (24).

Notice that if equation (24) holds, then \( r_j \) cannot be selected to increase \( C_j^p \) in the future. Equation (24) can be evaluated in constant time since the value of the sum term in it can be obtained in the previous step.

If equation (24) does not hold, the following condition is checked:
\[
4\delta + 2m - 2 \max \left\{ \lambda_{\max}, \lambda_i + \delta \right\} \leq m - 4\lambda_{\text{sum}}.
\] (26)

If equation (26) holds, then increasing \( C_i^p \) by \( \delta \) is feasible according to equation (6) and Theorem 2. Equation (26) can be checked in an incremental way; thus, the time required is \( O(1) \). If equation (26) does not hold, Algorithm 1 is applied. If no transaction can be selected in a step or every transaction’s preallocated computation time has reached its maximum value, the algorithm terminates and returns the current \( C_i^p \). The selection process is described in Algorithm 2.

More sophisticated optimization algorithms can be used for our optimization problem to obtain better solutions, such as the evolutionary algorithm and particle swarm optimization [36, 37]. However, as the number and the worst case computation times of sensor transactions can be large, these optimization algorithms may converge slowly. The usage of these algorithms for our optimization problem will be left as our future work. In addition, as will be illustrated in the experiment section, by applying the adaptive instance execution method on the transactions with preallocated computation times obtained from the greedy heuristic, the data quality can be greatly improved.

After determining the preallocated computation times and assigning transactions to processors, the system will execute the transaction instances to update the corresponding real-time data objects. In P-QM, when instance \( r_i \) arrives, if \( c_{i,j} \) is no larger than \( C_i^p \), then \( r_i \) is admitted, otherwise it is dropped. The admitted instances are scheduled using the EDF scheme.

5. The IP-QM Method

IP-QM also uses Algorithms 1 and 2 to determine \( C_i^p \) and assign transactions to processors. It further improves the data quality by executing transaction instances adaptively on each processor. Figure 1 shows the adaptive execution of instances on a processor.

As shown in Figure 1, when instance \( r_i \) arrives at processor \( p_s \), the instance dropper determines whether \( r_i \) is admitted or not. If \( r_i \)’s actual computation time \( c_{i,j} \) is no larger than \( C_i^p \), then \( r_i \) is admitted. Otherwise, the following condition is checked:
\[
\sum_{r_i, t_i \in s} \frac{c_i, j}{E(s)} + \sum_{r_i, t_i \in s} E(s) \cdot N(s, r_i) + c_{i,j} \leq D_i.
\] (27)

In equation (27), \( c_i, j \) is the total remaining computation time of \( r_i \)‘s instances with release times no larger than \( r_i \) and deadlines no larger than \( d_i,j \). \( d_i,j \) is the (absolute) deadline of \( r_i,j \), \( d_i,j = (I - 1)P_i + D_i \). \( E(s) \) is the average computation time of an instance of \( r_i \):
\[
E(s) = \int_{t \in \Gamma} t \cdot g_i(t) \, dt + Pr(t \geq C_i^p) C_i^p.
\] (28)

\( N(s, r_i) \) is the number of \( r_i \)’s instances with release times larger than \( r_i \) and deadlines no larger than \( d_i,j \):
\[
N(s, r_i) = \max \left\{ \left\lfloor \frac{d_i,j - D_i}{P_i} \right\rfloor - \frac{r_i, s}{P_i} + 1, 0 \right\}.
\] (29)

\( r_i \) is admitted if equation (27) is satisfied and is dropped otherwise. The remaining computation times of instances are obtained from the scheduler. \( E(s) \) can be precomputed in stage one. \( N(s, r_i) \) can be computed in constant time. Thus, the time required to check equation (27) is \( O(|\Gamma|) \). The check is only required when \( c_{i,j} > C_i^p \).

Two queues \( Q_p \) and \( Q_A \) are maintained for \( p_s \). The instances in each queue are arranged in non-decreasing order of their absolute deadlines. The admitted instance \( r_{i,j} \) is put into \( Q_p \) if \( c_{i,j} \leq C_i^p \). Otherwise, \( r_{i,j} \) is split into two instances: \( r_{i,j}^p \) with computation time \( C_i^p \) and \( r_{i,j}^p \) with computation time \( c_{i,j} - C_i^p \). \( r_{i,j}^p \) and \( r_{i,j}^p \) are put into \( Q_p \) and \( Q_A \), respectively. Both of them have the deadline \( D_i \). \( r_{i,j}^p \) can only be executed after \( r_{i,j}^p \) finishes. Each time the scheduler selects the instance at the head of \( Q_p \) for execution. If \( Q_p \) is empty, the instance at the head of \( Q_a \) is selected. An instance is aborted if it cannot be finished before its deadline.

In addition to the arriving instances, the instance dropper is also used to drop some instances that are already in the system. Let \( c_i, j(t) \) denote the computation time of the finished part of \( r_{i,j} \) at time instant \( t \). The following rules are used for dropping:

1. When \( r_{i,j} \) arrives, if \( r_{i,j+1} \) is not finished and \( c_{i,j+1}(r_{i,j}) + c_{i,j} \leq \min\{c_{i,j+1}, C_i^p\} \), then \( r_{i,j+1} \) is dropped. The deadline of \( r_{i,j} \) is set to be \( d_{i,j+1} \).
2. When \( r_{i,j} \) arrives, if \( r_{i,j}(\#) \)’s latest two instances \( r_{i,j} \) and \( r_{i,j-1} \) are not finished, \( r_{i,j-2} \) has been finished, \( d_{i,j} \leq d_{i,j-1} \) and \( c_{i,j-1}(r_{i,j}) + c_{i,j} \leq C_i^p + \max\{C_i^p - c_{i,j}, 0\} \), then \( r_{i,j-1} \) is dropped. The deadline of \( r_{i,j} \) is set to be \( d_{i,j-1} \). If \( c_{i,j} > C_i^p \), then \( r_{i,j}^p \) is removed from \( Q_A \) and put into \( Q_p \) with deadline \( d_{i,j-1} \).

For rule 2, if there are more than one transaction satisfying the dropping condition, the transaction with the largest skippable computation time, i.e., \( c_{i,j} - c_{i,j}(r_{i,j}) \), is selected for dropping. If \( C_i^p > c_{i,j} \), \( r_{i,j} \) will not be considered
Algorithm 2: Preallocated computation time determination.

as an dropping candidate in the future. The time required to evaluate the condition of rule 1 and rule 2 are $O(1)$ and $O(|\Gamma_k|)$, respectively.

The instance dropping does not affect the validity of data objects. Consider rule 1. If $\tau_{i,j-1}$ is dropped, then $\tau_{i,j}$ is certain to be finished before $d_{i,j-1}$. The corresponding data object $X_i$ is valid in the time interval $[t_1, d_{i,j+1}]$. $t_1$ is the finish time of $\tau_{i,j}$. If $\tau_{i,j-1}$ is not dropped and finished before $d_{i,j-1}$, then $X_i$ is valid in $[t_2, d_{i,j+1}]$. $t_2$ is the minimum of the finish time of $\tau_{i,j-1}$ and $\tau_{i,j}$. Otherwise, $X_i$ is valid in $[t_3, d_{i,j+1}]$. $t_3$ is the finish time of $\tau_{i,j}$. It can be seen that $t_2$ and $t_3$ are no smaller than $t_1$ since the remaining part of $\tau_{i,j-1}$ may be executed and $c_{i,j}$ is no larger than $c_{i,j-1} - c_{i,j-1}(\tau_{i,j})$. Consider rule 2. $X_i$ is valid from the finish time of $\tau_{i,j-3}$ to $d_{i,j-1}$. If $\tau_{i,j-1}$ is dropped, then the unused computation time of $\tau_{i,j}$, i.e., $C_{i,j}^p - c_{i,j}$ will be used by $\tau_{i,j}$. Thus, $\tau_{i,j}$ is certain to be finished before $d_{i,j-1}$ and $X_i$ is valid in $[d_{i,j-3}, d_{i,j+1}]$. If $\tau_{i,j-1}$ is not dropped, then $\tau_{i,j-1}$ and $\tau_{i,j}$ may not be finished before their deadlines if their actual computation times are larger than $C_{i,j}^p$. Thus, $X_i$ may be invalid in some intervals in $[d_{i,j-1}, d_{i,j+1}]$.

6. Experiment Evaluation

This section presents the results obtained from performance studies on the proposed methods.

The performance metrics used in the experiments are the average quality of individual data objects (ADQ\_IND), the average quality of correlated sets (ADQ\_COR), and the average update workload (AUW). The definition of ADQ\_IND and ADQ\_COR are given by equations (1) and (2). Let $W$ denote the simulation time and $L_p$ the total time that processor $\pi_p$ is busy executing the transaction instances:

$$AUW = \sum_{i \leq \text{processors}} L_pW / m.$$  

The parameters and default settings used in the experiments are presented in Table 2. The validity interval of a data object is uniformly distributed in $[2000, 4000]$. The computation time of a sensor transaction is generated following the normal distribution with mean computation time $EC_i$ and standard deviation $DC_i$ given in Table 2. $EC_i$ itself is uniformly selected in $[10, 20]$. The number of data objects in a correlated set is uniformly distributed in $[2, 8]$. These objects are randomly selected from the data object set. The threshold of a correlated set is set to be $N_k \cdot P_{th}$, where $P_{th}$ indicates the percentage of data objects in $Y_k$ that are permitted to be invalid at a time point.

Figure 2 shows the average quality of individual data objects when the number of sensor transactions varies from 100 to 200. The number of processors is 2. $P_{th}$ is set to be 0.4. When the number of transactions is no larger than 120, the ADQ\_IND of P-QM and IP-QM are equal to 1. Both methods’ ADQ\_IND decrease as the number of transactions increases. However, the ADQ\_IND of IP-QM keeps higher than that of P-QM. This is because for an incoming instance with actual computation time larger than the preallocated computation time, IP-QM will accept it for execution if equation (27) is satisfied, while P-QM always drops it. In addition, IP-QM will drop some instances that are already in
As discussed in Section 5, this kind of dropping does not affect the validity of data objects. However, it can leave more processor resources to instances with long execution times. Notice that the difference in the data quality between the two methods increases as the number of transactions increases. When the number reaches 200, the difference goes to about 0.37.

Figure 3 shows the average quality of correlated sets. It can be seen that the ADQ\_COR of IP-QM is constantly higher than that of P-QM. For example, when the number of transactions is 180, the ADQ\_COR of IP-QM is about 0.98, while the ADQ\_COR of P-QM is about 0.87. Both methods’ ADQ\_COR decrease with the increase in the number of transactions. However, IP-QM’s ADQ\_COR drops much slower than P-QM. One observation from Figures 2 and 3 is that the ADQ\_COR is higher than the ADQ\_IND under both methods. The reason is that the threshold of a correlated set is not large, which leads to a high probability that the correlated set is valid at a time point while the individual data objects in it are not. In addition, since the data objects in a correlated set are selected randomly from the whole data object set, the average valid ratio of them is very close to the average valid ratio of all data objects.

Figure 4 shows the average update workload. It can be seen that when the number of sensor transactions is larger than 120, the AUW of IP-QM is higher than that of P-QM. In addition, the AUW of IP-QM always goes up when the number of transactions increases, while the AUW of P-QM
goes up before the number reaches 140 and drops slowly after that. This is because in P-QM, when there are a large number of transactions in the system, to accommodate them, the preallocated computation times of them are decreased. This leads to more rejected instances which cancel out the increase in the system workload. In IP-QM, however, many of the instances rejected by P-QM can be accepted and finished due to the admission and dropping rules. These extra instances contribute to the higher average quality of individual data objects and correlated sets.

Figures 5 and 6 show the average quality of individual data objects and correlated sets under different setting of $P_{th}$, respectively. The number of sensor transactions is fixed at 200. It can be seen that the $ADQ_{IND}$ and the $ADQ_{COR}$ of IP-QM are higher than that of P-QM. For example, when $P_{th}$ is 0.3, the $ADQ_{IND}$ and the $ADQ_{COR}$ of IP-QM are about 0.87 and 0.92, while the $ADQ_{IND}$ and the $ADQ_{COR}$ of P-QM are about 0.53 and 0.58. There is not much changes in the $ADQ_{IND}$ of both methods since the average system workload under different values of $P_{th}$ are very close to each other when the number of transactions is fixed. Meanwhile, both methods’ $ADQ_{COR}$ increase as the value of $P_{th}$ increases. This is because a larger $P_{th}$ means fewer data objects in a correlated set are required to be valid; thus, there are more chances to access a correlated set that is valid under both methods.
Experiments were repeated for systems with 4 processors. Figures 7 and 8 show partial results. The number of transactions varies from 220 to 320. $P_{th}$ is set to be 0.4. It can be seen that IP-QM still outperforms P-QM in terms of $ADQ_{IND}$ and $ADQ_{COR}$. In addition, the performance degradation of both methods becomes much slower when more processor resources are available.

7. Conclusions
This paper studies the problem of data quality-based scheduling of sensor transactions on multiprocessor
platforms. A definition of data quality is given to describe the validity degree of real-time data objects. Two methods, P-QM and IP-QM, are proposed. P-QM maximizes the data quality by judiciously determining the preallocated computation times of sensor transactions and assigning the transactions to processors. IP-QM improves the data quality obtained from P-QM by adaptively executing transaction instances on each processor. Experiment results show that IP-QM outperforms P-QM in terms of the average quality of individual data objects and the average quality of correlated sets. In this work, the partitioned approach is adopted for transaction scheduling. In the future, we will study how to use the global approach and the hybrid approach to schedule the transactions so that the real-time data quality can be effectively maintained.

Data Availability
The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest
The authors declare that they have no conflicts of interest.

Acknowledgments
This research was supported by the Hunan Provincial Natural Science Foundation of China under Grant no. 2020JJ4032.

References
[1] M. Canizo, A. Conde, S. Charramendieta, R. Minon, R. G. Cid-Fuentes, and E. O nieva, “Implementation of a large-scale platform for cyber-physical system real-time monitoring,” IEEE Access, vol. 7, pp. 52455–52466, 2019.
[2] T. Liu, B. Tian, Y. Ai, and F.-Y. Wang, “Parallel reinforcement learning-based energy efficiency improvement for a cyber-physical system,” IEEE/CAA Journal of Automatica Sinica, vol. 7, no. 2, pp. 617–626, 2020.
[3] Y. Guo, X. Hu, B. Hu, J. Cheng, M. Zhou, and R. Y. K. Kwok, “Mobile cyber physical systems: current challenges and future networking applications,” IEEE Access, vol. 6, pp. 12360–12368, 2018.
[4] A. Miloslav and M. Veeraraghavan, “Sensor data fusion algorithms for vehicular cyber-physical systems,” IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 9, pp. 1762–1774, 2012.
[5] T. Bai, S. Li, and Y. Zheng, “Distributed model predictive control for networked plant-wide systems with neighborhood cooperation,” IEEE/CAA Journal of Automatica Sinica, vol. 6, no. 1, pp. 108–117, 2019.
[6] L. Qi, M. Zhou, and W. Luan, “A two-level traffic light control strategy for preventing incident-based urban traffic congestion,” IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 1, pp. 13–24, 2018.
[7] M. Xiong and K. Ramamritham, “Deriving deadlines and periods for real-time update transactions,” IEEE Transactions on Computers, vol. 53, no. 5, pp. 567–583, 2004.
[8] M. Xiong, S. Han, K.-Y. Lam, and D. Chen, “Deferrable scheduling for maintaining real-time data freshness: algorithms, analysis, and results,” IEEE Transactions on Computers, vol. 57, no. 7, pp. 952–964, 2008.
[9] M. Xiong, Q. Wang, and K. Ramamritham, “On earliest deadline first scheduling for temporal consistency maintenance,” Real-Time Systems, vol. 40, no. 2, pp. 208–237, 2008.
[10] J.-T. Wang, K.-Y. Lam, S. Han, S. H. Son, and A. K. Mok, “An effective fixed priority co-scheduling algorithm for periodic update and application transactions,” Computing, vol. 95, no. 10-11, pp. 993–1018, 2013.
[11] S. Han, K.-y. Lam, J. Wang, S. H. Son, and A. K. Mok, “Adaptive co-scheduling for periodic application and update transactions in real-time database systems,” Journal of Systems and Software, vol. 85, no. 8, pp. 1729–1743, 2012.
[12] S. Han, K.-Y. Lam, J. Wang, K. Ramamritham, and A. K. Mok, “On co-scheduling of update and control transactions in real-time sensing and control systems: algorithms, analysis and performance,” IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 10, pp. 2323–2342, 2012.
[13] M. Xiong, B. Liang, K.-Y. Lam, and Y. Quo, “Quality of service guarantee for temporal consistency of real-time transactions,” IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 8, pp. 1097–1110, 2006.
[14] A. Labrinidis and N. Roussopoulos, “Update propagation strategies for improving the quality of data on the web,” in Proceedings of the International Conference on Very Large Data Bases, pp. 391–400, Roma, Italy, September 2001.
[15] K.-D. Kang, S. H. Son, and I. A. Stankovic, “Managing deadline miss ratio and sensor data freshness in real-time databases,” IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 10, pp. 1200–1216, 2004.
[16] T. Gustafsson and J. Hansson, “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, IEEE, Toronto, Canada, May 2004.
[17] M. Amirijo, J. Hansson, and S. H. Son, “Specification and management of QoS in real-time databases supporting imprecise computations,” IEEE Transactions on Computers, vol. 55, no. 3, pp. 304–319, 2006.
[18] Y. Zhou and K.-D. Kang, “Deadline assignment and feedback control for differentiated real-time data services,” IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 12, pp. 3245–3257, 2015.
[19] K.-D. Kang, “Reducing deadline misses and power consumption in real-time databases,” in Proceedings of the IEEE Real-Time Systems Symposium, pp. 257–268, IEEE, Porto, Portugal, December 2016.
[20] L. Golab, T. Johnson, and V. Shkapenyuk, “Scalable scheduling of updates in streaming data warehouses,” IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 6, pp. 1092–1105, 2012.
[21] M. H. Bateni, L. Golab, M. T. Hajiaghayi, and H. Karloff, “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, Calgary, Canada, August 2009.
[22] W. Kang and J. Chung, “QoS management for embedded databases in multicore-based embedded systems,” Mobile Information Systems, vol. 2015, Article ID 657252, 2015.
[23] S. Ho, T. Kuo, and A. K. Mok, “Similarity-based load adjustment for real-time dataintensive applications,” in...
[24] M. Xiong, S. Han, D. Chen, K.-Y. Lam, and S. Feng, “DESH: overhead reduction algorithms for deferrable scheduling,” *Real-Time Systems*, vol. 44, no. 1-3, pp. 1–25, 2010.

[25] S. Han, D. Chen, M. Xiong, K.-Y. Lam, A. K. Mok, and K. Ramamritham, “Schedulability analysis of deferrable scheduling algorithms for maintaining real-time data freshness,” *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 979–994, 2014.

[26] J. Li, M. Xiong, V. C. S. Lee, L. Shu, and G. Li, “Workload-Efficient deadline and period assignment for maintaining temporal consistency under EDF,” *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1255–1268, 2013.

[27] G. Li, C. Zhou, J. Li, and B. Guo, “Maintaining data freshness in distributed cyber-physical systems,” *IEEE Transactions on Computers*, vol. 68, no. 7, pp. 1077–1090, 2019.

[28] X. Zhu, P.-C. Huang, S. Han, A. K. Mok, and M. Nixon, “MinMax: a sampling interval control algorithm for process control systems,” in *Proceedings of IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 68–77, IEEE, Seoul, South Korea, September 2012.

[29] A. Jha, M. Xiong, and K. Ramamritham, “Mutual consistency in real-time databases,” in *Proceedings of 27th IEEE International Real-Time Systems Symposium*, pp. 335–343, IEEE, Rio de Janeiro, Brazil, December 2006.

[30] S. Han, K.-Y. Lam, D. Chen et al., “Online mode switch algorithms for maintaining data freshness in dynamic cyber-physical systems,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 3, pp. 756–769, 2016.

[31] J. Li, J.-J. Chen, M. Xiong, G. Li, and W. Wei, “Temporal consistency maintenance upon partitioned multiprocessor platforms,” *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1632–1645, 2016.

[32] K. D. Kang, “Enhancing timeliness and saving power in real-time databases,” *Real-Time Systems*, vol. 54, no. 1, pp. 484–513, 2018.

[33] C. Zhou, G. Li, J. Li, and B. Guo, “Energy-Aware real-time data processing for IoT systems,” *IEEE Access*, vol. 7, pp. 171776–171789, 2019.

[34] I. D. Rob and B. Ala, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Computing Surveys*, vol. 43, no. 4, pp. 1–44, 2011.

[35] S. Baruhah and N. Fisher, “The partitioned multiprocessor scheduling of sporadic task systems,” in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 321–329, IEEE, Miami, FL, USA, December 2005.

[36] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, “A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems,” *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 875–887, 2019.

[37] Q. Wu, M. Zhou, Q. Zhu, Y. Xia, and J. Wen, “MOELS: multiobjective evolutionary list scheduling for cloud workflows,” *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 1, pp. 166–176, 2020.