Sharing-aware Data Acquisition Scheduling for Multiple Rules in the IoT

Seonyeong Heo
POSTECH
Pohang, Republic of Korea
heosy@postech.ac.kr

Seungbin Song
Yonsei University
Seoul, Republic of Korea
seungbin@yonsei.ac.kr

Bongjun Kim
POSTECH
Pohang, Republic of Korea
bong90@postech.ac.kr

Hanjun Kim
Yonsei University
Seoul, Republic of Korea
hanjun@yonsei.ac.kr

***Abstract*—In the Internet-of-Things (IoT) environments, users define event-condition-action (ECA) rules, and expect IoT frameworks to evaluate conditions and take appropriate actions within a certain time limit after an event occurs. To evaluate the conditions with fresh data items, the frameworks acquire required data from IoT sensors. Since the data acquisition causes battery consumption of sensors, the frameworks should minimize the number of the data acquisition while keeping the sensor data fresh until finishing the condition evaluation. However, existing data acquisition schedulers inefficiently acquire sensor data because the schedulers assume each ECA rule in a program is independent of each other although different rules may share some sensing data from the same sensors. This work proposes an efficient sharing-aware data acquisition scheduling algorithm that reduces unnecessary data acquisition by sharing sensor data commonly used in different rules while satisfying time constraints. To evaluate the proposed scheduling algorithm, this work deploys 19 devices in an office, collects values of 26 different sensors for 144 hours, and simulates the proposed algorithm and a baseline algorithm. Compared to the baseline algorithm, the proposed algorithm reduces communication count and deadline miss ratio by 31.9% and 50.2% respectively.**

I. Introduction

Popular Internet-of-Things (IoT) frameworks such as SmartThings [1] and IFTTT [2] allow users to define event-condition-action (ECA) rules with their IoT devices that specify which action should be taken in a certain condition when an event occurs. Fig. 1 illustrates a smart office example. In the example, a user constructs a smart office environment based on rules like "When motion is detected (event), if the illuminance intensity is less than 50 lux (condition), then turn on the light (action)". To make the rules materialize, an IoT server monitors events, acquires data from sensors, evaluates the conditions, and sends commands to actuating devices.

According to the rules, a user expects the IoT server to take actions within a time limit after an event occurs. When the IoT server detects an event occurrence, the server should evaluate conditions of the rules associated with the event, and decide whether to take appropriate actions or not. Since the conditions compare data items from IoT sensors and threshold values, the server should acquire fresh data items from the sensors to reflect the real-time user environment. For example, the condition of Rule 1 in Fig. 1 requires the humidity value from the humidity sensor and the illuminance value from the light sensor. When motion is detected, the server should request fresh humidity and illuminance values to the humidity and light sensors, and evaluate the condition. Here, although prefetching the data items before the event occurrence can reduce the data acquisition overheads and response time, the prefetching is infeasible especially for sporadic events such as human motion, because it is difficult to predict when the events will occur. Moreover, since IoT sensors drain their batteries to measure the user environment and send the sensing data, the number of the data requests to sensors should be minimized to save sensor batteries. Therefore, efficient data acquisition scheduling is important to give a prompt response to users with fresh data while minimizing the number of data requests.

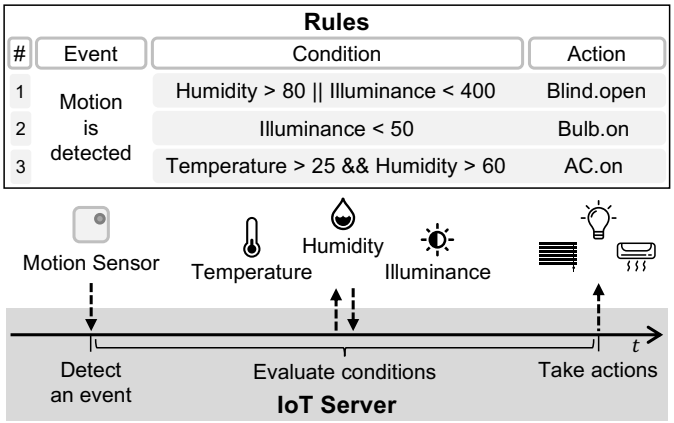
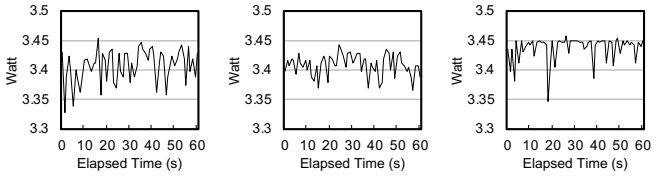


Fig. 1: Smart office example

To reduce the data acquisition overheads, data acquisition scheduling needs to be aware of data item sharing among different rules. Since users have a finite number of IoT devices and write rules using the data items of the devices, different rules may share the same data item from the same device. In other words, different rules can be correlative in terms of data items they use. In the smart office example in Fig. 1, Rule 1 and Rule 2 share the illuminance value, and Rule 1 and Rule 3 share the humidity value. In these cases, the IoT server does not need to acquire the illuminance and humidity values twice to evaluate two different conditions. Therefore, considering sharing data items among different rules in data acquisition scheduling can reduce unnecessary data acquisitions and enable prompt response to an event.

Previous work [3]–[7] has studied data acquisition schedul-



(a) Period = 1s (b) Period = 100ms (c) Period = 10ms

Fig. 2: Energy consumption with different data acquisition periods

ing in decision-making problems and proposes scheduling strategies to find an optimal acquisition order of data items. However, they assume each decision task (condition evaluation in the context of this work) is independent of each other. None of them primarily consider data sharing among decision tasks, causing unnecessary communication cost for acquiring shared data items. Although Kim et al. [5] propose a heuristic to skip acquiring a shared data item that is already acquired by the other task, the scheduling strategy itself is not aware of data item sharing, so the strategy does not actively reduce duplicate data acquisitions for the shared data items. Therefore, the proposed solutions cannot successfully increase data sharing when different rules share the same data items.

This work proposes an efficient sharing-aware data acquisition scheduling algorithm for multiple rules, S-tLVF (*Sharing-aware tree-based Least Volatile item First*), that reflects data item sharing in the scheduling. Basically, S-tLVF schedules data acquisitions for each rule in parallel and adjusts the original schedule to avoid unnecessary data acquisitions. For a single rule, this work introduces the tLVF algorithm that schedules data acquisitions by calculating the expected costs of acquiring data items with considering the nested structure of a rule condition, because the state-of-the-art data acquisition scheduling algorithm [3] only supports a restricted form of a rule condition. S-tLVF schedules data acquisitions for each rule using tLVF, and tries to delay reacquiring (already) fresh data items. This work also proposes a predictive S-tLVF algorithm that delays data acquisition schedule to avoid unnecessary data acquisition and enables data sharing among different rules in the future.

This work evaluates the proposed scheduling algorithms by deploying 19 IoT sensors and devices in an office room and collecting 26 different kinds of data items for 144 hours. This work generates two test cases: 1) 10 events and 200 rules with all the 26 data items (a large test case) and 2) 10 events and 200 rules with only 10 of the 26 data items (a small test case). The work simulates the proposed algorithms and a baseline algorithm [3]. Compared to the baseline algorithm, S-tLVF reduces data acquisition counts and deadline miss ratio by 31.9% and 50.2% for the large case and by 34.7% and 48.3% for the small case, respectively. S-tLVF obtains the better performance with the small case because rules are more likely to share the same data items. The evaluation results show that S-tLVF successfully reduces the number of unnecessary data acquisitions and enables more rules to meet their deadlines.

The contributions of this paper are:

- Efficient sharing-aware data acquisition scheduling algorithm (S-tLVF) for multiple rules in the IoT environments
- Predictive sharing-aware data acquisition scheduling (PS-tLVF) that schedules data acquisitions to realize sharing opportunities in the future
- Evaluation with real-world IoT data, collected on the smart office testbed

II. Background & Motivation

A. Data Acquisition in IoT Frameworks

IoT frameworks such as SmartThings [1], Microsoft Flow [8] and IFTTT [2] allow users to import existing rules and write custom rules that follow event-condition-action ways. When an event occurs, the framework server acquires a set of data items from IoT devices in the user environments, evaluates the conditions of the rules, and executes the actions if the conditions are satisfied. For example, Fig. 1 illustrates that a user installs three rules in an IoT framework. According to Rule 1, when motion is detected, the server acquires the humidity and illuminance values from IoT sensors, and opens the blind if the room is humid or dark. Here, since acquiring data items requires round trip communication between the server and IoT devices, data acquisition scheduling is a crucial key to quickly and efficiently evaluate the conditions.

Along with the quality of services, efficient data acquisition can also enhance the energy efficiency of IoT devices. In general, acquiring data items less times results in less energy consumption. To demonstrate the correlation, this work measures the energy consumption of an Odroid C2 compute board when a client requests a data item periodically. Fig. 2 shows how the energy consumption changes as the data acquisition period increases. The result implies that if the client requests a data item more frequently, the energy consumption tends to increase. Therefore, efficient data acquisition scheduling is important to save the battery of IoT devices.

However, complex time constraints in the IoT pose challenges to find an efficient data acquisition scheduling strategy. In data acquisition scheduling, IoT frameworks need to consider four different aspects: deadlines of rules, early termination of condition evaluation, freshness intervals of date items, and data item sharing among rules.

First, each rule has its own *deadline*. Users expect IoT frameworks to take appropriate actions within a certain time limit after an event occurs [4], [5], [7], [9]. For example, users may want the frameworks to turn on bulbs when they enter a room according to Rule 2 in Fig. 1. If the action is delayed, users have to unpleasantly turn on the bulbs on their own. Therefore, the IoT frameworks should acquire data items, evaluate the conditions, and take the actions within the deadline.

Second, IoT frameworks can terminate condition evaluation without acquiring all data items in rules (*early termination of condition evaluation*). The IoT frameworks may acquire all the data items in parallel at the same time to meet the

deadline. However, it causes unnecessary data acquisition and energy consumption of IoT sensors because the frameworks can evaluate a condition with a partial set of the data items. For Rule 1 in Fig. 1, if the humidity value is larger than 80 or if the illuminance value is smaller than 400, the framework can decide to open the blind without acquiring the other value. Therefore, data acquisition scheduling needs to consider which data item is more likely to terminate the condition evaluation early to minimize unnecessary data acquisition.

Third, each data item has its own *freshness (validity)* interval, a period in which the data item can remain valid. Since IoT sensors measure user environments that are ever changing, the sensor values are valid only within a certain period. During the condition evaluation, if a data item becomes invalid after its freshness interval, the IoT frameworks should acquire the data item from the sensors again to refresh the sensor value. Moreover, since some sensor values fluctuate more than others, their freshness intervals are different. For example, some sensor values such as temperatures and humidity slowly change while the others like motion and illuminance may quickly change over time. Then, the freshness intervals of the former ones would be relatively larger than those of the latter ones. Therefore, the IoT frameworks should reflect the freshness intervals of sensors in data acquisition scheduling.

Finally, data acquisition scheduling should reflect *data item sharing among different rules*. Since a limited number of IoT devices engage in multiple rules in the user environment, different rules are highly likely to share the same data items. For example, in Fig. 1, both Rule 1 and Rule 2 use the illuminance value, and both Rule 1 and Rule 3 use the humidity value in their conditions. Therefore, the rules are not independent but correlative.

To demonstrate that many real-world applications actually share data items, this work analyzes 70 open-source SmartThings applications at SmartThings Community [10] that use commodity IoT devices. This work collects all the rules that subscribe to a single event or multiple events in the applications, and then this work counts shared data items among the rules. Here, to see the maximum possibility of data sharing, this work assumes that the same type of data items belong to one IoT device although actual applications may use different devices for the same type of data items. The analysis results show that most applications share some data items with other applications. Among the 70 applications, 91% of the applications share some data items with other applications, and 61% of the applications share more than one data items. Since different applications and rules can reuse the shared data items, considering data sharing in data acquisition scheduling can reduce unnecessary communications with IoT devices.

B. Limitation of Existing Scheduling Schemes

Previous work [3]–[7] investigates data acquisition scheduling for decision-making problems. Hu et al. [3] find an optimal data acquisition policy for a single rule, which chooses a data item with the longest freshness interval first. Note that this work refers the single-rule policy as LVF (*Least Volatile item First*) instead of its original name (LDF; *Least Deadline First*) to avoid confusion. Based on LVF, Hu et al. propose a data acquisition scheduling algorithm for multiple rules that reflects early termination of condition evaluation to reduce unnecessary data acquisitions. The algorithm calculates short-circuiting probabilities of each data item and gradually modifies the LVF order.

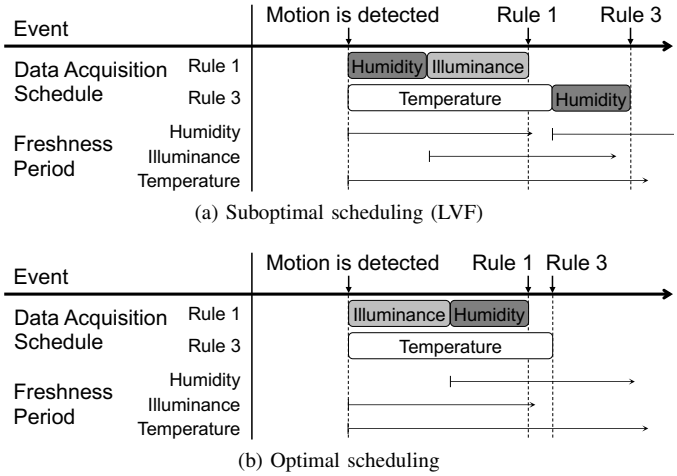


Fig. 3: Motivating timeline graphs

Extending the previous work [3], Kim et al. [5] propose EDEF-LVF (*Earliest Deadline or Expiration First - Least Volatile item First*), an optimal data retrieval policy for multiple rules. EDEF-LVF chooses a rule with the earliest deadline and freshness expiration first. Then, it schedules to acquire data items one by one for the chosen rule using the LVF policy.

However, existing data acquisition algorithms [3]–[7] have limitations in scheduling real-world IoT applications. First, the algorithms ignore data sharing among different rules and then incur redundant data acquisitions. Since the algorithms assume that rules are independent of each other, the algorithms may not obtain the best performance when different rules share the same data items. Second, the scheduling algorithms do not fully reflect early termination of condition evaluation, which can largely affect the communication efficiency. Although Hu et al. [3] consider the short-circuiting probabilities of each data item, they assume that each condition consists of a conjunction of multiple sub-conditions restricting the semantics of rules.

Fig. 3 illustrates a motivating example that demonstrates the need to consider data sharing among different rules. In Fig. 3, Rule 1 and Rule 3 refer to Rule 1 and Rule 3 in Fig. 1. Each horizontal bar indicates the data acquisition latency of a data item, and each arrow under the horizontal bars indicates the freshness interval of a data item. Assuming that none of the data items (Temperature, Humidity, Illuminance) are valid when motion is detected, the IoT framework should acquire Temperature, Humidity, Illuminance to evaluate the conditions of Rule 1 and Rule 3. Without considering the correlations between Rule 1 and Rule 3, the suboptimal scheduling, LVF, decides to acquire Humidity first for Rule 3 because the freshness interval of Humidity is longer than the freshness interval of Illuminance. However, the optimal

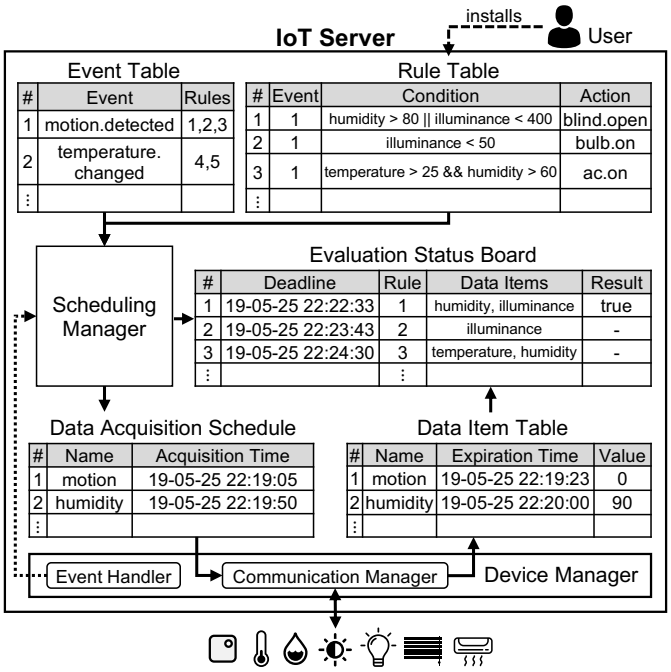


Fig. 4: The overall design of the IoT framework

scheduling is to acquire Illuminance first considering that Rule 1 and Rule 3 can share the value.

Following the observations, considering data sharing among different rules is important to reduce redundant data acquisitions and response time of IoT rules. Although real-world IoT applications do share the same data items, existing scheduling schemes assume that rules are independent of each other. Therefore, the schemes cannot always find optimal solutions. To overcome their limitations, this work proposes efficient data acquisition scheduling algorithms that consider data item sharing among different rules.

III. Overall Design of the IoT Framework

This section describes the overall design of an IoT framework that this work is based on. The IoT framework assumes that a central IoT server integrates all IoT devices. The IoT server consists of five major data structures: an event table, a rule table, an evaluation status board, a data acquisition schedule, and a data item table, and two major modules: a device manager and a scheduling manager. Fig. 4 briefly illustrates the overall framework.

The IoT framework allows users to install event-condition-action rules that describe which action is required in a condition when an event occurs. The syntax of rules can be expressed in a context-free language as described in Fig. 5. An event represents a notification from an IoT device. The IoT device can push the notification to the server, or the server can pull the device status and notice the event from the status change. A condition of a rule is a binary expression that determines when to take actions according to data items values. Actions of a rule describe how actuator devices act in response to the event. Note that unlike the existing frameworks, the

$$
\begin{aligned}
\text{rule} &::= \texttt{WHEN}\ \text{event} \\
&\quad\ \texttt{IF}\ \text{conditions} \\
&\quad\ \texttt{THEN}\ \text{actions} \\
&\quad\ \texttt{WITH}\ \text{constraint} \\
\text{event} &::= \textit{Device.event} \\
\text{conditions} &::= (\text{conditions}\ \texttt{\&\&}\ \text{conditions}) \\
&::= (\text{conditions}\ \texttt{||}\ \text{conditions}) \\
&::= \text{condition} \\
\text{condition} &::= v \triangle k \\
\triangle &::= \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{==} \mid \texttt{!=} \\
\text{actions} &::= \text{actions; action} \\
&::= \text{action} \\
\text{action} &::= \textit{Device.operation()} \\
\text{constraint} &::= \texttt{TIME} \leq D
\end{aligned}
$$

Fig. 5: The syntax of an event-condition-action rule. Here, v is a data item from an IoT device, and k is a user-defined threshold value.

framework allows users to specify a relative deadline of a rule as a time constraint. If users do not specify any, the framework sets a default value for the time constraint. When the event occurs, the framework should evaluate the condition by checking whether the condition is satisfied or not using fresh data items within the deadline.

When a user installs a new rule, the server stores the rule in the rule table. Each entry in the rule table is linked with an event in the event table. Each entry in the event table contains information about an event to subscribe and rules to trigger when the event occurs. If a rule introduces a new event, then the framework inserts a new entry into the event table and links the entry with the rule. If a new rule subscribes to the existing event, then the framework links the existing event entry with the rule.

The device manager communicates with IoT devices to acquire data items and detect events. The device manager consists of an event handler and a communication manager. The event handler communicates with IoT devices to detect events and notifies the scheduling manager to reschedule data item acquisitions. The communication manager acquires data items from the devices on data acquisition schedule, and sends appropriate commands to actuators if the conditions of the rules are satisfied. Here, the communication manager measures network latencies to each device, and keeps their distribution to find their worst-case network latencies for the current network environment.

When an event occurs, the scheduling manager finds out which rules to trigger based on the event table. First, when an event occurs, the scheduling manager inserts new entries for the triggered rules into the evaluation status board which contains deadlines, data items to acquire, and condition evaluation status. Then, the scheduling manager determines when to acquire data items for the rules waiting to be evaluated. Finally, the scheduling manager updates the data acquisition schedule. Here, the framework does not prefetch the required data items because events are sporadic and unpredictable in

TABLE I: Notation for the problem

Notation	Description
R	Set of rules associated with an event
r_i	The i-th rule in R
D_r	Relative deadline of Rule r
V_r	Set of data items to acquire for Rule r
v_r^i	The i-th data item in V_r
t_r	Time when Rule r is triggered
t_r^E	Time when the evaluation of Rule r ends
$I[v]$	Freshness interval of Data item v
$L[v]$	Data acquisition latency of Data item v

the IoT.

When the IoT server receives a data item, the communication manager calculates the expiration time of the data item with its freshness interval, and updates its value and expiration time in the data item table. The framework evaluates conditions with the data item and other fresh data items. If the framework terminates the condition evaluation of a rule, the framework updates the evaluation result in the evaluation status board. If the result is true, the framework instructs the communication manager to send an action command to the target device. Finally, the scheduling manager erases the rule from the evaluation status board.

IV. Problem Statement and Solution

This section examines a data acquisition scheduling problem when the IoT framework in Section III evaluates conditions with shared data items. Section IV-A formulates the problem using the notation in Table I, and Section IV-B introduces scheduling algorithms as solutions.

A. Problem Statement

A rule consists of an event, conditions and actions as defined in Fig. 5. The event describes a situation that triggers the rule. If the event that Rule r subscribes occurs at t_0, Rule r will be triggered at t_0 (i.e., $t_r = t_0$). The conditions describe environmental conditions in which the actions to be taken. Each condition uses data items that sensors collect from the user environment. In Fig. 1, Rule 1 uses the humidity and illuminance data items to describe an environmental condition to open the blind. This work formulates the set of data items that Rule r uses as V_r.

Each data item has a different freshness interval and data acquisition latency. Freshness interval is a period in which the data item can remain valid after the last update. After the freshness interval from the last update, the data item becomes invalid (no longer fresh). Then, the framework should acquire the data item again to use its value for evaluation. Data acquisition latency is the time required to pull data from a physical device. This work uses the notation of $I[v]$ and $L[v]$ to represent the freshness interval and the data acquisition latency of Data item v.

Let R be a set of the rules to evaluate. To evaluate the conditions of rules in R, the framework plans to acquire data items in $\bigcup_{r \in R} V_r$. Note that the conditions of two different rules can share the same data items. In other words, there can be a pair of data items $v_{r_1} \in V_{r_1}$ and $v_{r_2} \in V_{r_2}$ such that $v_{r_1} = v_{r_2}$ where $r_1 \neq r_2$. In the scheduling process, to reduce data acquisition overheads while satisfying the deadlines of rules, the framework determines which data item to acquire first. This work formulates the problem as follows.

Problem Statement. The problem is to find a data acquisition schedule $S = ((v_1, t_1), (v_2, t_2), ..., (v_m, t_m))$ that satisfies the following constraints where v_i is a data item to acquire and t_i is the time to acquire the data item for $i = 1, ..., m$:

1) **Deadline constraints**: Each rule's condition evaluation has finished before its deadline, i.e., for every $r \in R$,
$$t_r^E \leq t_r + D_r$$
2) **Data freshness constraints**: All the necessary data items are fresh when each rule's condition evaluation finishes, i.e., for every $r \in R$ and $v \in V_r$, there exists $(v, t) \in S$ such that
$$t_r^E \leq t + I[v]$$

For solving the problem, this work uses several assumptions: (i) A rule cannot be triggered when the rule is under evaluation. (ii) The worst-case latency for acquiring a data item is given by the IoT framework for the current network condition. Note that different data items can have different acquisition latencies. (iii) Relative deadline of each rule and a freshness interval of each data item are given in advance. (iv) The processing time for choosing a rule and ordering data items is negligible compared to data acquisition latencies.

B. Solution

This section describes data acquisition scheduling algorithms as solutions for the problem in Section IV-A. First, this work introduces a single-rule scheduling algorithm (tLVF) that orders data items considering the nested structure of a rule condition. Second, this work proposes a sharing-aware scheduling algorithm (S-tLVF) for multiple rules that avoids reacquiring shared data items. Then, this work extends the sharing-aware scheduling algorithm to increase data sharing by examining more data sharing chances (PS-tLVF).

Basically, this work schedules data acquisitions for each respective rule in parallel and modifies the schedule to increase data sharing. Unlike EDEF-LVF [5] which schedules all data acquisitions in a sequential order, this work schedules data acquisitions for each rule in parallel based on two premises: (i) for a single rule, sequential processing is generally more efficient than parallel processing because condition evaluation may terminate early with a partial set of data items, and (ii) for multiple rules, sequential processing is generally less efficient than parallel processing with fewer chances of data sharing due to data freshness constraints.

tLVF (tree-based LVF): To schedule data acquisitions for a single rule, this work introduces tLVF (Algorithm 1) by

Algorithm 1: tLVF, which finds an efficient acquisition order of data items for a single condition

Input : A condition tree T of a rule
A set of data items V to acquire
Output : An order of data items $O = (v_1, v_2, ...)$

1 $O_v \leftarrow$ Sort $v \in V$ in the LVF order
2 **while** $O_v \neq \emptyset$ **do**
3 $(p, L_e) \leftarrow \texttt{ExpLatency}(T, O_v)$
4 $L \leftarrow$ Sort $(v, l) \in L_e$ in ascending order of l
5 **for** $(v, l) \in L_e$ **do**
// $\oplus$: concatenation operator
6 $O_{tmp} \leftarrow O \oplus \{v\} \oplus (O_v \setminus \{v\})$
7 **if** O_{tmp} *meets freshness constraints* **then**
8 $O_v \leftarrow O_v \setminus \{v\}$
9 $O \leftarrow O \oplus \{v\}$
10 break
11 **end**
12 **end**
13 **end**

Algorithm 2: ExpLatency, which calculates the expected condition evaluation latency when each data item is the next item to acquire

Input : A condition tree T of a rule
A set of data items V to acquire
Output : The probability p that T becomes true
A list of pairs of data item and expected latency L_e

1 **if** $T = \textbf{Leaf}(v, \triangle, k)$ **then**
2 **if** $v \in V$ **then**
3 **if** v *is fresh* **then**
4 $(p, L_e) \leftarrow (v \triangle k, \{(v, 0)\})$
5 **else**
6 $(p, L_e) \leftarrow (P(V_v \triangle k), \{(v, L[v])\})$
7 **end**
8 **else**
// v is already scheduled
9 $(p, L_e) \leftarrow (null, \emptyset)$
10 **end**
11 **else if** $T = \textbf{Internal}(T_l, bop, T_r)$ **then**
12 $(p_l, L_l) \leftarrow \texttt{ExpLatency}(T_l, V)$
13 $(p_r, L_r) \leftarrow \texttt{ExpLatency}(T_r, V)$
14 **if** $L_l = \emptyset$ **then**
// the left child is already scheduled
15 $(p, L_e) \leftarrow (p_r, L_r)$
16 **else if** $L_r = \emptyset$ **then**
// the right child is already scheduled
17 $(p, L_e) \leftarrow (p_l, L_l)$
18 **else**
19 $l_l^{min} \leftarrow \texttt{min}(\{l | (v, l) \in L_l\})$
20 $l_r^{min} \leftarrow \texttt{min}(\{l | (v, l) \in L_r\})$
21 **if** $bop = \&\&$ **then**
22 $(p, L_e) \leftarrow (p_l * p_r,$
$\{(v, l + p_l * l_r^{min}) | (v, l) \in L_l\} \cup$
$\{(v, l + p_r * l_l^{min}) | (v, l) \in L_r\})$
23 **else if** $bop = ||$ **then**
24 $(p, L_e) \leftarrow (1 - (1 - p_l) * (1 - p_r),$
$\{(v, l + (1 - p_l) * l_r^{min}) | (v, l) \in L_l\} \cup$
$\{(v, l + (1 - p_r) * l_l^{min}) | (v, l) \in L_r\})$
25 **end**
26 **end**
27 **end**

extending the existing algorithm [3] to support the nested structure of a rule condition. The existing algorithm is not applicable to the problem, because it only considers a conjunction of comparison expressions like $(v_1 < k_1)\&\&(v_2 < k_2)\&\&...\&\&(v_m > k_m)$. On the other hand, tLVF allows the nested structure of a rule condition in the form of a tree data structure [9], where each leaf node represents a single comparison expression and each internal node connects two child trees with a binary operator. Then, a condition tree T can be defined as

$$T = \textbf{Internal}(T_{left}, bop, T_{right}) \mid \textbf{Leaf}(v, \triangle, k)$$

where $bop \in \{\&\&, ||\}$.

tLVF schedules the data acquisition order reflecting the nested structure of a rule condition and the probability that its evaluation will terminate early. First, tLVF calculates the original LVF order (a data item with the longest freshness interval first) for given the data items in V and their freshness intervals (Line 1). Then, to increase the probability of the early termination, tLVF invokes the ExpLatency function to calculate the minimum expected condition evaluation latencies for each data item (Line 3). tLVF adjusts the LVF order by putting a data item with the lowest expected latency first if not violating data freshness constraints (Line 7). Once some data items are scheduled, tLVF recalculates the expected latencies reflecting the scheduled data items at the next iteration of the outer loop.

ExpLatency (Algorithm 2) calculates the expected latencies of each data item when the IoT framework acquires the data item next. Since tLVF may not choose the data item with the lowest latencies first due to the freshness constraints, ExpLatency calculates the minimum expected latencies for all the data items in V assuming that tLVF chooses each data item first.

To calculate the expected latencies, ExpLatency recursively traverses the condition tree. For a leaf node, ExpLatency returns the probability that the leaf node condition is true and the expected data acquisition latency. If the data item is fresh, ExpLatency evaluates the condition with the acquired value and forces the expected latency of the data item as 0 (Line 4). If the data item is previously scheduled, ExpLatency returns the empty set of expected latencies, not to affect the other parts of the condition (Line 9). For an internal node, ExpLatency calculates both cases when tLVF chooses the next data item from the left or right children. For example, if bop is $\&\&$ and tLVF chooses the next item from the left child, the condition evaluation may terminate if the evaluation result of the left child is false, or continue to evaluate the right child. Thus, the expected latency becomes the sum of the expected latency of the chosen data item (l) and the minimum expected latency of the right child ($p_l * l_r^{min}$). ExpLatency calculates the other case also and unions the results of the both cases.

The complexity of tLVF is $O(|V|^3)$ which is same with that of the existing algorithm [3]. Since each ExpLatency recursively invokes itself twice in a divide-and-conquer manner,

Algorithm 3: S-tLVF, schedules data acquisitions while avoiding reacquiring shared data items

```
   Input : Data acquisition schedule S = {(v, t, r), ...}
1  if an event occurs or a data acquisition is due then
2      R_t ← A set of triggered rules
3      for r ∈ R_t do
4          O_r ← tLVF(V_r)
5          t_tmp ← t_r
6          for v ∈ O_r do
7              S ← S ∪ (v, t_tmp, r)
8              t_tmp ← t_tmp + L[v]
9          end
10     end
11     for (v, t, r) ∈ S such that t = t_cur do
12         if t + L[v] < ExpireAt[v] then
               // delay the data item until expiring
13             Replace (v, t, r) with (v, ExpireAt[v] − L[v], r)
14             if v was already delayed once then
15                 continue
16             end
               // advance the other data items
17             for (v', t', r') ∈ S such that (v' ≠ v) ∧ (r' = r) do
18                 Replace (v', t', r') with (v', t' − L[v], r')
19             end
20         else
21             Acquire Data item v
22         end
23     end
24 end
```

its complexity is $O(|V|^2)$ in the worst case. The complexity of checking freshness constraint violations (Line 7 in Algorithm 1) is $O(|V|)$, and the inner loop iterates $O(|V|)$ times. Therefore, the complexity of tLVF is $O(|V|^3)$.

S-tLVF (Sharing-aware tLVF): On top of tLVF, this work proposes S-tLVF (Algorithm 3) that reduces unnecessary data acquisitions by delaying reacquisition of fresh data items until the data items expire. When an event occurs, S-tLVF schedules data acquisitions of triggered rules with tLVF. Then, if a data item to acquire is fresh, S-tLVF delays acquiring the data item. After delaying the data acquisition schedule, S-tLVF acquires the other data items in advance while keeping the tLVF order. In Algorithm 3, ExpireAt contains the expiration time of data items.

Theorem 1. *S-tLVF preserves the deadline satisfaction of the original schedule by tLVF.*

S-tLVF modifies the original schedule by delaying fresh data items and advancing the other data items for each rule.

Lemma 1.1. *In S-tLVF, delaying fresh data items preserves the deadline satisfaction of original schedule.*

Proof. Let r be the rule in interest and t_r^E be the evaluation time of Rule r when acquiring data items with the original schedule. Assume that S-tLVF delays the acquisition of Data item v from t to t' for Rule r.

- *Case i)* $t_r^E < t + L[v]$:
  Since the evaluation of Rule r finishes while Data item v is fresh, delaying acquiring Data item v does not change the evaluation time of Rule r.
- *Case ii)* $t_r^E \geq t + L[v]$:
  Based on the hypothetical assumption that the original schedule satisfies the deadline constraint, so the evaluation must finish before Data item v expires, i.e.,

$$t_r^E \leq t + I[v]$$

  After delaying, the expiration time of Data item v lengthens from $t + I[v]$ to $t' + I[v]$, so t_r^E will not change.

Therefore, delaying the acquisitions of fresh data items will not change t_r^E and preserves the deadline satisfaction. □

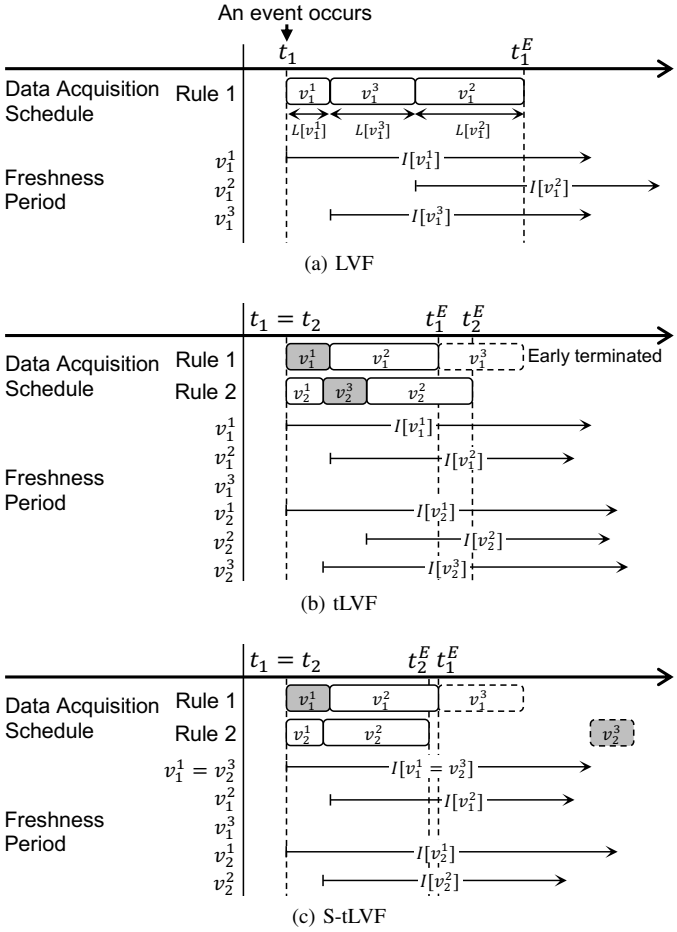


Fig. 6: Timeline examples for LVF, tLVF, and S-tLVF

Lemma 1.2. *In S-tLVF, advancing the other data items preserves the deadline satisfaction of original schedule.*

Proof. Proof by contradiction.

Assume that the evaluation misses the deadline with the new schedule. Let V_r the set of necessary data items for evaluating Rule r. Additionally, let t'_v and t'^E_r be the times when the framework acquires Data item $v \in V_r$ and when the evaluation finished with the new schedule, respectively.

Due to the assumption,

$$t'^E_r > t^E_r$$

Algorithm 4: ForwardSharing, which delays acquiring a shared data item of which acquisition is undergoing

Input : Data acquisition schedule $S_r = \{(v_1, t_1), ...\}$ for Rule r
Undergoing data acquisition schedule $S_u = \{(v_1, t_1), ...\}$
Maximum delay t_d^{max} allowed for Rule r
Output : t_d^f: A delay for forward sharing

$V_a \leftarrow \{v|(v,t) \in S_r\}$, $V_u \leftarrow \{v|(v,t) \in S_u\}$
$(v_1, t_1) \leftarrow$ the earliest data acquisition in S_r
$t_d^f \leftarrow 0$
for $v \in V_a \cap V_u$ **do**
  $t_v \leftarrow$ the arrival time of v in S_u
  $t_d^{tmp} \leftarrow t_v - t_1$
  if $t_d^{tmp} \leq t_d^{max}$ **then**
    $t_d^f \leftarrow \texttt{max}(t_d^f, t_d^{tmp})$
  end
end

S-tLVF advances data acquisition time of the other items, and thus

$$t'_v \leq t_v$$

Since the evaluation finishes when the framework acquires the last data item, for the original and new schedules,

$$t_r^E = \texttt{max}(\{t_v + L[v] | v \in V_r\})$$
$$t_r'^E = \texttt{max}(\{t'_v + L[v] | v \in V_r\})$$

Because $t'_v \leq t_v$,

$$\texttt{max}(\{t'_v + L[v] | v \in V_r\}) \leq \texttt{max}(\{t_v + L[v] | v \in V_r\})$$
$$t_r'^E \leq t_r^E$$

Since there is a contradiction with the assumption, advancing the other data items preserves the deadline satisfaction of original schedule in S-tLVF. □

Proof. With Lemma 1.1 and Lemma 1.2, delaying fresh data items and advancing the other data items in S-tLVF preserve the deadline satisfaction of the original schedule. Therefore, S-tLVF preserves the deadline satisfaction of the original schedule.

□

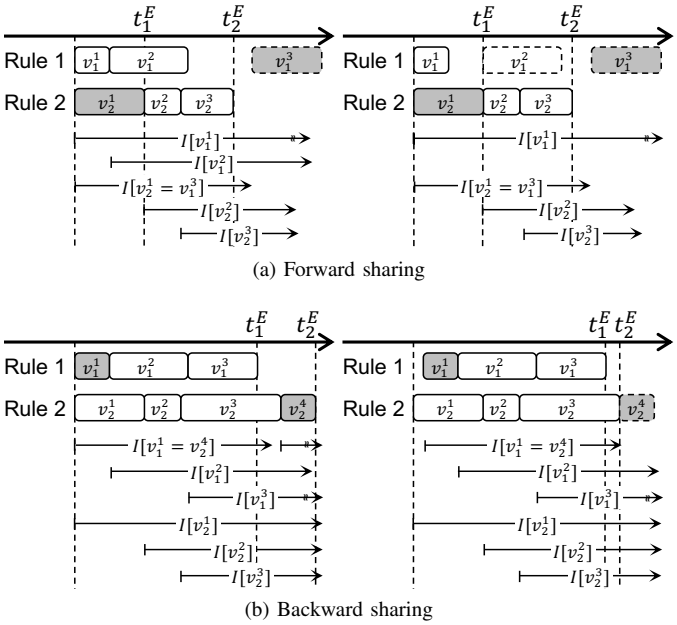
Fig. 6 shows the timeline graphs when LVF, tLVF, and S-tLVF schedule data acquisitions for two rules (Rule 1 and Rule 2). LVF acquires the data item with the longest freshness interval first without considering the nested structure nor shared data items (Fig. 6(a)). On the other hand, tLVF reflects the nested structure of conditions in the scheduling. Although v_1^3 has a longer freshness interval than v_1^2, tLVF schedules v_1^2 earlier than v_1^3 and increases the chance of early evaluation termination (Fig. 6(b)). S-tLVF schedules data items considering data sharing among rules. Since Rule 1 and Rule 2 can share v_2^3, S-tLVF delays reacquiring v_2^3 and acquires v_2^2 in advance. Advancing the acquisition schedule of v_2^2 shortens the evaluation termination time, t_2^E (Fig. 6(c)).

The computational complexity of S-tLVF is $O(|R||V|^3)$. Since the complexity of tLVF is $O(|V|^3)$, and S-tLVF executes

Algorithm 5: BackwardSharing, which delays acquiring a shared data item considering future data sharing

Input : Rule r to test backward sharing
Original data acquisition schedule $S = \{(v_1, t_1, r_1), ...\}$
Maximum delay t_d^{max} allowed for Rule r
Output : A delay t_d^b required for backward sharing
A rule r^b that enables backward sharing

$V_a \leftarrow \{v' | r' = r, (v', t', r') \in S\}$
$(v_1, t_1, r) \leftarrow$ the earliest data acquisition for Rule r in S
$t_d^b \leftarrow 0$
for $(v', t', r') \in S$ **do**
  if $(v' = v_1) \wedge (t' > t_1 + I[v_1]) \wedge (r' \neq r)$ **then**
    if $\exists (v'', t'', r'') \in S, (v'' \in V_a) \wedge (t'' < t') \wedge (r'' = r')$ **then**
      continue
    end
    $t_d^{tmp} \leftarrow t' - (t_1 + I[v_1])$
    if $(t_d^{tmp} \leq t_d^{max}) \wedge \{(t_d^b = 0) \vee (t_d^{tmp} < t_d^b)\}$ **then**
      $(t_d^b, r_b) \leftarrow (t_d^{tmp}, r')$
    end
  end
end



(a) Forward sharing

(b) Backward sharing

Fig. 7: Examples of forward and backward data sharing

tLVF for each rule (Line 5 in Algorithm 3), its complexity becomes $O(|R||V|^3)$. Although S-tLVF adjusts the original schedule to reduce redundant data acquisitions (Line 11 to Line 23), its complexity is $O(|R||V|^2)$ only. Therefore, the complexity of S-tLVF is $O(|R||V|^3)$.

PS-tLVF (Predictive S-tLVF): To enlarge the coverage of the S-tLVF algorithm, this work proposes PS-tLVF that extends S-tLVF to consider future data sharing. Although S-tLVF is simple and effective, there still exist additional opportunities that can reduce unnecessary data acquisitions. First, the scheduler can delay acquiring a shared data item if another rule is acquiring the data item at the moment (forward sharing, Algorithm 4). Since the condition evaluation

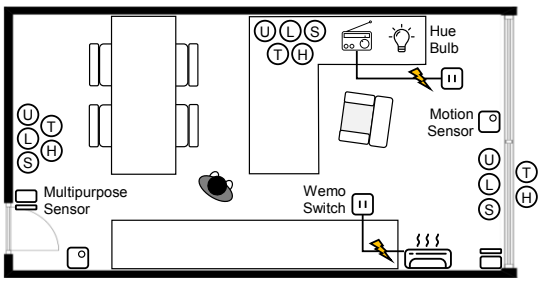


Fig. 8: Smart office testbed (Ⓣ, Ⓗ: Temperature-humidity sensor, Ⓤ: Ultrasonic sensor, Ⓛ: Light sensor, Ⓢ: Sound sensor)

may terminate early by the shared item, forward sharing can reduce unnecessary data acquisitions. Second, the scheduler can delay acquiring a shared data item to make the data item remain fresh until other rules start to acquire the data item (backward sharing, Algorithm 5). In the both cases, PS-tLVF conservatively delays data acquisitions by setting the maximum time delay t_d^{max} as the gap between the minimum time constraint (t_c^{min}) and the last acquisition time for a rule:

$$t_c^{min} \leftarrow \texttt{min}(\{t_r + D_r\} \cup \{\mathsf{ExpireAt}(v) | v \in V_r^f\} \cup \{t + I[v] | (v, t, r) \in S_u\})$$

where V_r^f is the set of fresh data items in V_r and S_u is the undergoing data acquisition schedule.

Fig. 7 shows how PS-tLVF additionally reduces unnecessary data acquisitions compared with S-tLVF. For Fig. 7a and Fig. 7b, the left and right timeline graphs show the schedules by the S-tLVF and PS-tLVF algorithms, respectively. Fig. 7(a) shows an example case of forward sharing. With S-tLVF, the framework finishes the condition evaluation of Rule 1 after acquiring $\{v_1^1, v_1^2\}$. However, PS-tLVF delays the acquisition of v_1^2, allows condition evaluation to finish with $\{v_1^1, v_1^3 = v_2^1\}$, and reduces the unnecessary acquisition of v_1^2. Fig. 7(b) shows an example case of backward sharing. Considering that Rule 2 requires $v_2^4 = v_1^1$, delaying the acquisition of v_1^1 and subsequent data acquisitions can reduce the unnecessary acquisition of v_2^4.

V. Evaluation

This work evaluates how the proposed algorithms in Section IV-B reduce data acquisition counts and deadline misses by simulating the framework model in Section III with real-world IoT data. To obtain realistic IoT data, this work constructs a smart office testbed with 19 IoT devices listed in Table II. In the table, FI and AL are abbreviations of 'Freshness Interval' and 'Data Acquisition Latency'. As aggregators, this work deploys three Raspberry Pi 3 boards. Each board has a temperature-humidity sensor, an ultrasonic sensor, a light sensor, and a sound sensor. This work implements a SmartThings [1] application to catch all the events on the SmartThings-compatible devices. Fig. 8 shows the floor plan of the smart office testbed with the positioning of the IoT devices.

TABLE II: Device specification of the testbed

Device	#	Data Item	Type	Range	FI (s)	AL (s)
SmartSense Motion Sensor	2	Motion	Binary	[0,1]	7	3
		Temperature	Integer	[5,30]	30	3
SmartSense Multipurpose Sensor	2	Contact	Binary	[0,1]	10	3
		Temperature	Integer	[5,30]	30	3
Temperature-Humidity Sensor	3	Temperature	Integer	[5,30]	30	1
		Humidity	Integer	[10,60]	30	1
Ultrasonic Sensor	3	Distance	Float	[0,500]	5	1
Light Sensor	3	Light level	Integer	[0,700]	10	1
Sound Sensor	3	Sound level	Integer	[200,500]	5	1
Philips Hue	1	Switch	Binary	[0,1]	15	5
Belkin Wemo Switch	2	Switch	Binary	[0,1]	15	5

On the testbed, this work collects real-time values of 26 data items for 144 hours with one agent. The agent performs several prescribed behaviors in the office such as opening the window or turning on the radio, and the devices observed the behaviors. Although some IoT devices provide multiple data items, the server acquires data items from the same device individually in the simulation. We have made our smart office dataset publicly available via GitHub [11].

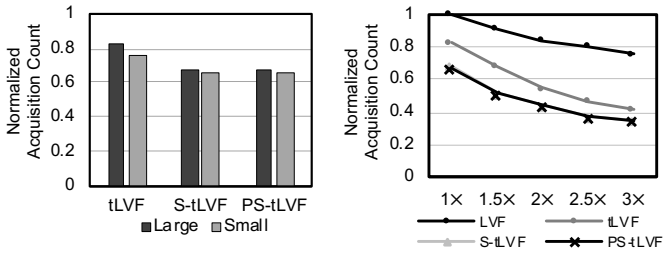
A. Methods

Using the collected data on the smart office testbed, this work simulates the framework model in Section III with the proposed algorithms. To show that the proposed scheduling algorithms effectively schedule data item acquisitions for evaluating multiple rules, this work compares the scheduling algorithms with a baseline scheduling algorithm in terms of deadline miss and data acquisition count:

- **LVF (Least Volatile item First)**: schedules each rule in parallel by choosing a data item with the longest freshness interval first.
- **tLVF (tree-based Least Volatile item First)**: schedules each rule in parallel with the tLVF algorithm (Algorithm 1).
- **S-tLVF (Sharing-aware tree-based Least Volatile item First)**: schedules each rule in parallel with the tLVF algorithm and adjusts the original schedule to avoid reacquiring shared data items (Algorithm 3).
- **PS-tLVF (Predictive Sharing-aware tree-based Least Volatile item First)**: optimizes S-tLVF with forward and backward sharing (Algorithm 4 and 5) to further reduce unnecessary data acquisitions by considering more data sharing cases.

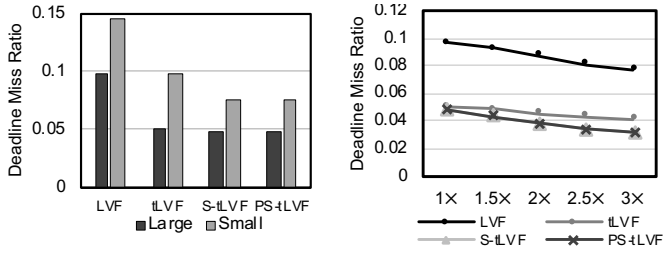
For the algorithms based on tLVF, this work uses the normal distributions of data item values in the dataset to calculate the probability that a condition becomes true.

In the evaluation, this work specifies freshness intervals and data acquisition latencies for data items as in Table II. This work generates two test cases such as a large test case



(a) Different test cases (b) Different freshness intervals

Fig. 9: Normalized data acquisition count



(a) Different test cases (b) Different freshness intervals

Fig. 10: Deadline miss ratio

that uses all the 26 data items and a small test case that uses only 10 of the 26 data items. This work also manually specifies 10 events that a rule can subscribe. In addition, This work automatically generates 200 rules by randomly choosing values for defining a rule within the predefined range of data item values. For example, the condition value for a humidity item is chosen between 10 to 60 based on the collected data. Each rule condition is in a conjunctive normal form, which is a conjunction (the *and* operator '&&') of clauses. Each clause is a single comparative expression or a disjunction (the *or* operator '||') of comparative expressions. Therefore, skipping some data acquisitions is possible when one of the clauses is false or one of the comparative expressions in a clause is true. Here, the number of data items used in a rule varies from 1 to 10. For each rule, this work randomly sets the relative deadline considering the total latency of acquiring all the data items.

This work analyzes data sharing opportunities in the two test cases. First, each rule shares at least one data item with 105 and 154 other rules on average in the large and small test cases, respectively. Among the rules that subscribe to the same event, each rule shares at least one data item with 10 and 15 other rules on average in the large and small test cases, respectively. Second, each rule shares 1.5 and 2.0 data items on average with another rule in the large and small test cases, respectively. For example, Rule 73 (sound.desk $>$ 335 || temperature.door $<$ 16) and Rule 79 (motion.door $=$ none && sound.desk $<$ 293) share one data item such as the sound level around the desk in the large test case.

When an event occurs, the simulator starts to schedule data acquisitions for the rules that the event triggers. Each event can occur at multiple times during the simulation. Although the simulator schedules the same rules for the same event, the scheduling results can be different across different event occurrences because the results of condition evaluation may differ according to data items values at the moment. In other words, the framework may terminate the condition evaluation early according to data item values.

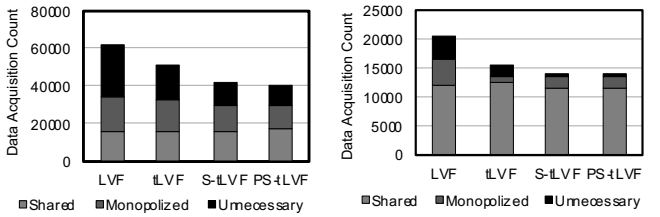
During the simulation, this work measures data acquisition counts and deadline miss ratios. The data acquisition count is the total number of communications between the server and devices, and the deadline miss ratio is the ratio of missed rules to the total triggered rules.

B. Results

Data acquisition count: For LVF, tLVF, S-tLVF, and PS-tLVF, this work measures data acquisition counts with the large and small test cases. Fig. 9(a) shows the normalized acquisition counts of tLVF, S-tLVF, and PS-tLVF to LVF. Compared with LVF, tLVF reduces the number of data acquisitions by 17.2% and 24.9%, and S-tLVF reduces the number by 31.9% and 34.7% for the large and small test cases respectively. Here, for the small test case, S-tLVF reduces data acquisition counts more than the large test case because each rule shares more data items with others in the small test case than the large test case. Thus, if a user installs more and more rules in an IoT environment, there will be more chances for data sharing, and S-tLVF will become more effective. PS-tLVF further reduces the number of data acquisitions by 33.1% and 34.7% compared with LVF for the large and small test cases respectively. It is because PS-tLVF can find the chances of forward and backward sharing for large test case but not for small case.

Fig. 9(b) shows how the normalized data acquisition counts of LVF, tLVF, S-tLVF, and PS-tLVF change as scaling the freshness intervals of data items for the large test case. The numbers on the x-axis indicate the multipliers applied to the original freshness intervals. This work normalizes each total data acquisition count to the total data acquisition count of the original freshness intervals ($1\times$). The evaluation result shows that the total acquisition count decreases as the freshness intervals increase because data items are more likely to be shared with longer freshness intervals for all the algorithms. Moreover, the result also shows that S-tLVF and PS-tLVF send less data acquisition requests than LVF and tLVF regardless of the freshness intervals.

Deadline miss ratio: This work measures the deadline miss ratio of LVF, tLVF, S-tLVF, and PS-tLVF for the large and small test cases. Fig. 10(a) shows the deadline miss ratio of LVF, tLVF, S-tLVF, and PS-tLVF. Compared with LVF, tLVF reduces the deadline miss ratio by 48.0% and 32.4%, and S-tLVF reduces the deadline miss ratio by 50.2% and 48.3% for the large and small test cases, respectively. The results show that tLVF and S-tLVF do not only reduce the data acquisition counts, but also reduce the deadline misses as Theorem 1 implies. With the forward and backward sharing algorithms, PS-tLVF slightly reduces the deadline miss ratio by 50.2%



(a) Large (b) Small

Fig. 11: Data acquisition count breakdown

and 48.3% compared with LVF for the large and small test cases, respectively. By exploring more sharing cases, PS-tLVF finds more chances of early termination than S-tLVF, and thus slightly reduces unnecessary data acquisitions and deadline misses especially for the large test case.

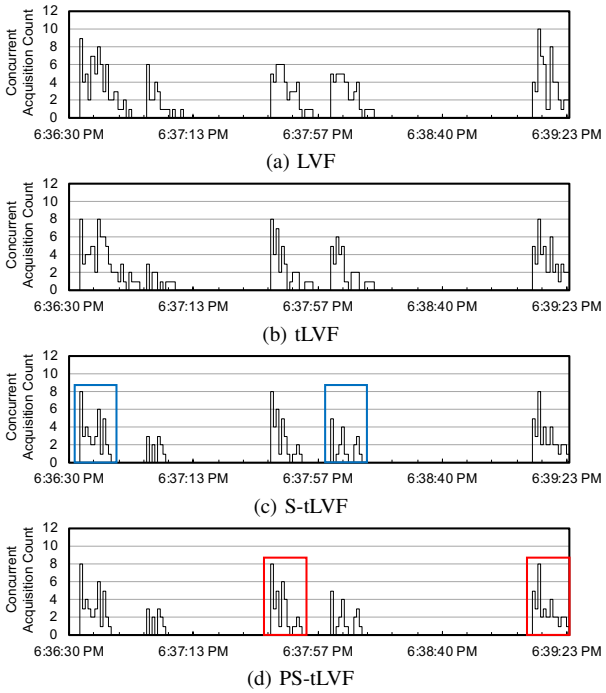
Fig. 10(b) shows how the deadline miss ratio of LVF, tLVF, S-tLVF, and PS-tLVF changes as scaling the freshness intervals of data items for the large test case. The evaluation result shows that the miss ratio tends to decrease as the freshness intervals increase for all the algorithms. For LVF, it is obvious to have more chances that acquired data items are still fresh at the moment of condition evaluation. For S-tLVF and PS-tLVF, it is because that the longer freshness intervals give the more chances for S-tLVF and PS-tLVF to find sharable data items and meet deadlines by the early termination.

Optimality: This work evaluates the optimality of the scheduling algorithms in terms of data acquisition count. This work measures how many unnecessary data acquisitions still remain for each scheduling algorithm. Fig. 11 shows the breakdown of data acquisitions over the total data acquisitions. In the figure, 'Unnecessary' indicates data acquisitions never used in evaluating rules. Note that 'Unnecessary' includes data acquisitions that cannot contribute to the early termination of rules. Similarly, 'Monopolized' and 'Shared' indicate data acquisitions used in evaluating a single rule or multiple rules.

As shown in Fig. 11, S-tLVF and PS-tLVF successfully reduce the amount of unnecessary data acquisitions compared with LVF. In other words, S-tLVF and PS-tLVF increase data sharing among rules. However, S-tLVF and PS-tLVF still have unnecessary data acquisitions. Here, since early termination is determined after acquiring data items, finding an optimal acquisition order that leads to the most early termination of condition evaluation is almost infeasible.

Data acquisition pattern: This work analyzes how the data acquisition patterns of the scheduling algorithms differ by measuring concurrent data acquisition counts for each scheduling algorithm. During the simulation, the maximum number of concurrent data acquisitions is 14, 14, 10, and 10 for LVF, tLVF, S-tLVF, and PS-tLVF, respectively. This result implies that S-tLVF and PS-tLVF can be more efficient than LVF and tLVF in terms of network bandwidth usage.

Fig. 12 illustrates concurrent data acquisition counts for a specific period. The results show that the acquisition patterns of the scheduling algorithms are different. Blue and red boxes indicate the periods where S-tLVF and PS-tLVF reduce



(a) LVF

(b) tLVF

(c) S-tLVF

(d) PS-tLVF

Fig. 12: Concurrent data acquisition count

data acquisition counts in the evaluation, respectively. The most remarkable point is that S-tLVF and PS-tLVF delay the original schedule to avoid unnecessary data acquisitions, showing flatter patterns than tLVF in Fig. 12.

VI. Discussion

Though S-tLVF and PS-tLVF successfully reduce unnecessary data acquisitions, S-tLVF and PS-tLVF cannot guarantee that data sharing is maximal. The optimal scheduling varies depending on the actual values of data items due to early termination. Since S-tLVF and PS-tLVF schedule data items to acquire data items without knowing the values, the algorithms are suboptimal. However, this work can extend the algorithms to consider the probability of early termination of other rules when deciding whether to share a data item or not.

This work also leaves several interesting issues in data acquisition scheduling as future work. First, this work assumes that all the rules are equally important. However, the priority of rules can differ according to their semantics. For example, some rules related to safety (e.g., fire alarm) may be more important than the others. Second, this work notices race conditions can exist among rules. Depending on the data arrival order from multiple sensors, rule evaluation time can change. It can cause unnecessary data acquisitions, but it will not cause constraint violations.

VII. Related Work

Data acquisition scheduling with freshness constraints: Scheduling algorithms for decision-making problems [3]–[5] find an optimal ordering of data acquisitions to make decisions before deadlines if each data item has a different freshness interval. Scheduling data acquisitions to meet both freshness

and deadline constraints is important to provide real-time decision making.

To meet the real-time constraints, previous work proposes optimal scheduling for a single decision task [3], [4] or multiple decision tasks [5]. However, previous work regards the decision tasks are independent of each other although they may share the same data items. If a currently fresh data item will not expire until another task that shares the same data item finishes, reacquiring the data item for the task is unnecessary. To increase data acquisition efficiency, this work proposes a scheduling algorithm that is aware of data item sharing while satisfying the real-time constraints.

In the domain of real-time database systems, the freshness of a data item is one of the important factors that affect the quality of query processing [12]. Previous work [13]–[15] focuses on data updates of real-time databases to meet freshness and deadline constraints. Adelberg et al. [13] point out that the real-time database should balance data freshness and transaction deadline, and suggest a transaction scheduling method to keep data fresh without violating the deadline. Lee et al. [14] propose periodic, aperiodic, and on-demand update schemes to meet freshness and time constraints, and compare the complexity and performance of their schemes. Xiong et al. [15] defer the sampling time (update period) of transactions as late as possible to reduce the number of updates and processor workload while preserving the temporal validity of data and meeting relative deadlines.

RTEDBS [16] and QeDB [17] are embedded real-time databases that aim to meet both deadline constraints and data freshness constraints of queries on resource-constrained embedded systems. These database systems dynamically adjust the amount of workloads and size of buffer cache between I/O devices and CPU to meet deadline constraints and reduce I/O overheads. While previous work does not change the data acquisition order for given queries, this work can reduce I/O overheads by modifying the data acquisition order.

Data sharing for multi-query optimization: To process multiple queries efficiently, previous work has proposed multi-query optimization techniques for traditional database systems [18]–[20] and wireless sensor networks [21]–[24].

Roy et al. [18] propose methods to reduce duplicated computations by extracting common sub-expressions from multiple queries and reusing the computed result. Dalvi et al. [19] suggest methods to pipeline the evaluation of a sub-expression to reduce multiple evaluation of common sub-expressions and duplicated acquisition of the results of the sub-expressions. Instead of extracting common sub-expressions from multiple queries, MiniTasking [20] exploits data sharing among concurrent queries to improve temporal locality by batching queries and scheduling operators. However, the previous work does not consider data freshness.

Based on an acquisitional query processing system such as TinyDB [25], [26], previous work [21]–[24] applies multi-query optimizations into wireless sensor networks. Trigoni et al. [21] suggest algorithms that remove duplicated and unnecessary data transmission from multiple queries by considering the topology of installed sensors. Müller and Alonso [22] propose algorithms that merge multiple user queries into a network query and decide common data sampling rate of the user queries. Xiang et al. [23], [24] propose Two-Tier Multiple Query Optimization (TTMQO) that calculates benefits of merging queries and eliminates duplicated data transmissions by considering shared structures of wireless sensor networks. Although previous work optimizes multiple queries to reduce communication and energy consumptions considering resource-constrained sensor networks, they only focus on how to eliminate redundant operations without concerning data freshness that affects the quality of query responses.

Prediction-based data acquisition scheduling: Value prediction can be useful to optimize data acquisition or query scheduling by estimating a data value without retrieving the actual value. Previous work [6], [7], [9], [27]–[31] has proposed real-time data acquisition systems that exploit value prediction.

Recent publications [6], [7], [9], [31] have focused on scheduling data acquisitions in the rule-based IoT systems using sensor value prediction. RT-IFTTT [9] is a real-time IoT framework that dynamically changes polling intervals of sensors by predicting whether a condition will be satisfied or not after a certain interval. If the framework expects a condition is likely to be satisfied soon, it shortens the polling interval to meet deadline constraints. Also, there have been frameworks that exploit logical contexts [31] or boolean expressions of conditions [6], [7] to reduce evaluation costs with sensor value prediction. If a context or a condition is likely to be true with the prediction, the evaluation of other contradicting contexts or conditions may be skippable. By applying the methods [6], [7], [9], [31], this work can improve the proposed scheduling algorithms.

Prediction-based data acquisition scheduling has also been widely discussed in sensor networks and real-time databases [27]–[30]. BBQ [27] and Ken [28] are database systems that optimize query paths and decide the order of sensors to visit in multi-hop sensor networks with sensor value prediction; the former adopts pull-based acquisitions, and the latter allows push-based event detection. Other data acquisition scheduling schemes [29], [30] calculate trade-offs of using predictions in terms of communication and energy costs and adaptively decide prediction schemes and sensors to observe.

VIII. Conclusion

In the IoT, different rules can share the same data items; in other words, the conditions of different rules are correlated in terms of data item acquisition. However, previous work on data acquisition scheduling assumes the condition evaluation of each rule is independent of each other. This work proposes S-tLVF, a sharing-aware data acquisition scheduling algorithm that efficiently evaluates correlated tree-structured conditions in multiple rules within a predefined deadline. Using the collected data in the read-world IoT environment, this work shows that S-tLVF reduces the number of data acquisition requests and deadline miss ratio by 31.9% and 50.2% respectively compared to the previous work.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers and shepherd for their valuable feedback. This research was supported by NRF-2015M3C4A7065646, NRF-2017R1C1B3009332, IITP-2017-0-00195 and IITP-2018-0-01392 through the National Research Foundation of Korea (NRF) and the Institute of Information and Communication Technology Planning and Evaluation (IITP) funded by the Ministry of Science and ICT. Hanjun Kim is the corresponding author of this paper.

REFERENCES

[1] "SmartThings," http://www.smartthings.com.
[2] "IFTTT," https://ifttt.com.
[3] S. Hu, S. Yao, H. Jin, Y. Zhao, Y. Hu, X. Liu, N. Naghibolhosseini, S. Li, A. Kapoor, W. Dron, L. Su, A. Bar-Noy, P. Szekely, R. Govindan, R. Hobbs, and T. F. Abdelzaher, "Data Acquisition for Real-Time Decision-Making under Freshness Constraints," in *2015 IEEE Real-Time Systems Symposium*, 2015, pp. 185–194.
[4] J. E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, R. Hobbs, and W. Dron, "On Maximizing Quality of Information for the Internet of Things: A Real-Time Scheduling Perspective (Invited Paper)," in *Proceedings of the IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications*, 2016, pp. 202–211.
[5] J. E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, and R. Hobbs, "Sporadic Decision-Centric Data Scheduling with Normally-off Sensors," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 135–145.
[6] T. Abdelzaher, M. T. A. Amin, A. Bar-Noy, W. Dron, R. Govindan, R. Hobbs, S. Hu, J. E. Kim, J. Lee, K. Marcus, S. Yao, and Y. Zhao, "Decision-Driven Execution: A Distributed Resource Management Paradigm for the Age of IoT," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1825–1835.
[7] J. Lee, K. Marcus, T. Abdelzaher, M. T. A. Amin, A. Bar-Noy, W. Dron, R. Govindan, R. Hobbs, S. Hu, J.-E. Kim, L. Sha, S. Yao, and Y. Zhao, "Athena: Towards Decision-Centric Anticipatory Sensor Information Delivery," *Journal of Sensor and Actuator Networks*, vol. 7, no. 1, 2018.
[8] "Microsoft Flow," https://flow.microsoft.com.
[9] S. Heo, S. Song, J. Kim, and H. Kim, "RT-IFTTT: Real-Time IoT Framework with Trigger Condition-Aware Flexible Polling Intervals," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, 2017, pp. 266–276.
[10] "SmartThings Comunity," https://community.smartthings.com/c/projects-stories/created-smart-apps.
[11] "Smart Office Dataset," https://github.com/corelab-src/smart-office-dataset.
[12] K. Ramamritham, "Real-time Databases," *Distributed and Parallel Databases*, vol. 1, no. 2, pp. 199–226, April 1993.
[13] B. Adelberg, H. Garcia-Molina, and B. Kao, "Applying Update Streams in a Soft Real-time Database System," in *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 1995, pp. 245–256.
[14] C.-G. Lee, Y.-K. Kim, S. H. Son, S. L. Min, and C. S. Kim, "Efficiently Supporting Hard/Soft Deadline Transactions in Real-Time Database Systems," in *Proceedings of 3rd International Workshop on Real-Time Computing Systems and Applications*, 1996, pp. 74–80.
[15] M. Xiong, S. Han, and K.-Y. Lam, "A Deferrable Scheduling Algorithm for Real-Time Transactions Maintaining Data Freshness," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, 2005, pp. 11–37.
[16] W. Kang, S. H. Son, J. A. Stankovic, and M. Amirijoo, "I/O-Aware Deadline Miss Ratio Management in Real-Time Embedded Databases," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 277–287.
[17] W. Kang, S. H. Son, and J. A. Stankovic, "QeDB: A Quality-Aware Embedded Real-Time Database," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2009, pp. 108–117.
[18] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe, "Efficient and extensible algorithms for multi query optimization," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00, 2000, pp. 249–260.
[19] N. N. Dalvi, S. K. Sanghai, P. Roy, and S. Sudarshan, "Pipelining in multi-query optimization," in *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '01, 2001, pp. 59–70.
[20] Y. Zhang, Z. Chen, and Y. Zhou, "Minitasking: Improving cache performance for multiple query workloads," in *Proceedings of the 7th International Conference on Advances in Web-Age Information Management*, ser. WAIM '06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 287–299.
[21] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman, "Multi-query optimization for sensor networks," in *Proceedings of the First IEEE International Conference on Distributed Computing in Sensor Systems*, ser. DCOSS'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 307–321.
[22] R. Muller and G. Alonso, "Efficient sharing of sensor networks," in *2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, Oct 2006, pp. 109–118.
[23] S. Xiang, H. B. Lim, and K.-L. Tan, "Impact of multi-query optimization in sensor networks," in *Proceedings of the 3rd Workshop on Data Management for Sensor Networks: In Conjunction with VLDB 2006*, ser. DMSN '06, 2006, pp. 7–12.
[24] S. Xiang, H. B. Lim, K. Tan, and Y. Zhou, "Two-tier multiple query optimization for sensor networks," in *27th International Conference on Distributed Computing Systems (ICDCS '07)*, June 2007, pp. 39–39.
[25] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03, 2003, pp. 491–502.
[26] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: An acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005.
[27] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven Data Acquisition in Sensor Networks," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, 2004, pp. 588–599.
[28] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong, "Approximate Data Collection in Sensor Networks Using Probabilistic Models," in *Proceedings of the 22Nd International Conference on Data Engineering*, 2006, pp. 48–59.
[29] B. Gedik, L. Liu, and P. S. Yu, "ASAP: An Adaptive Sampling Approach to Data Collection in Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 12, pp. 1766–1783, Dec 2007.
[30] H. Jiang, S. Jin, and C. Wang, "Prediction or Not? An Energy-Efficient Framework for Clustering-Based Data Collection in Wireless Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 1064–1071, June 2011.
[31] S. Nath, "ACE: Exploiting Correlation for Energy-efficient and Continuous Context Sensing," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, 2012, pp. 29–42.