IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 20, 2020, accepted April 30, 2020, date of publication May 6, 2020, date of current version May 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2992512

Design of Smart Home Implementation Within IoT Natural Language Interface

TAE-YEUN KIM[1], SANG-HYUN BAE[2], AND YOUNG-EUN AN[3]
[1]SW Convergence Education Institute, Chosun University, Gwangju 61452, South Korea
[2]Department of Computer Science and Statistics, Chosun University, Gwangju 61452, South Korea
[3]College of General Education, Chosun University, Gwangju 61452, South Korea

Corresponding authors: Sang-Hyun Bae (shbae@chosun.ac.kr) and Young-Eun An (yean@chosun.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (MSIT), Korea Government, under Grant 2019R1F1A1041186.

ABSTRACT To process continuous sensor data in Internet of Things (IoT) environments, this study optimizes queries using multiple MJoin operators. To achieve efficient storage management, it classifies and reduces data using a support vector machine (SVM) classification algorithm. A global shared query execution technique was used to optimize multiple MJoin queries. By comparing each kernel function of the SVM classification algorithm, the system's performance was evaluated through experiments according to the selected optimal kernel function and changes in sliding window size. Furthermore, to implement a smart home system that can actively respond to users, classified and reduced sensor data were utilized to enable the intelligent control of devices inside the home. The sensor data (e.g., temperature, humidity, gas) used to recognize the current conditions of an IoT-based smart home system and corresponding date data were classified into decision trees, and the system was designed using five sensors to intelligently control priorities such as ventilation, temperature, and fire and intrusion detection. The experiments demonstrated that the multiple MJoin technique yields high improvements in performance with relatively few searches. In this study, the sigmoid was selected as the optimal kernel function for the SVM classification algorithm. According to the SVM classification algorithm results, based on changes in the sliding window size, the average error rate was 2.42%, the reduction result was 17.58%, and the classification accuracy was 85.94%. According to the comparison of the classification performance of SVM and other algorithms, the SVM classification algorithm exhibited a minimum 9% better classification performance. Thus, compared to existing home systems, this algorithm is expected to increase system efficiency and convenience by enabling the configuration of a more intelligent environment according to the user's characteristics or requirements.

INDEX TERMS Application, Internet of Things (IoT), sensor data, smart home system, SVM algorithm.

I. INTRODUCTION

The purpose of the modern Internet of Things (IoT) is to provide services based on intelligent systems considering user convenience and accuracy. Researchers are conducting various studies on creating smart environments, such as smart homes, smart grids, and industrial IoT environments [1]. Typically, IoT-based smart home technologies include IoT features that go beyond conventional network technologies. The devices within the home automatically establish relationships with other devices, to improve domestic lifestyle services [2], [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Honghao Gao.

A variety of IoT technologies are required to construct intelligent systems in locations that require continuous management, such as IoT-based smart home systems, or to support remote control and monitoring services and create environments that are suitable to the user through communications between devices.

In particular, as the degree of sensor usage increases, a technology that processes real-time sensor data is considered a necessary component of an IoT environment to provide services that are customized for the user. Because various types of IoT data in an IoT environment are nonlinear and inconsistent owing to the time-series data characteristics and variety of sensor detection methods, the data have atypical properties [4]. Numerous sensors are needed to collect data in an

IoT environment, and an efficient processing method for the sensor data collected through a sensor network is necessary [5]. In a conventional home IoT system, the user changes the environment manually by controlling smart devices that are connected to the same network via a mobile device [6], [7]. However, recent IoT-based home systems use a method that operates devices intelligently according to the threshold values defined to control the environment.

Such IoT-based home systems must intelligently select devices to perform tasks according to changes in the environmental data within the home, and they must transmit warning messages to the user before dangerous situations occur within the home, so that the user can deal with them [8].

However, conventional IoT-based home systems operate passively according to the changes in environmental data within the home, and therefore the user must manually select devices by themselves. Because user intervention cannot be freely performed, it is difficult to change a device's status at the desired time. For example, sensors with low rates of use can lead to data waste and power loss when tasks are executed according to rules set by the developer without accounting for specific characteristics of a user [9]–[11].

This study uses multiple MJoin operators to efficiently process sensor data (stream data) in an IoT environment. A global shared query execution technique was used for query optimization, and the SVM classification algorithm was used to classify and reduce the data to enable efficient storage management. Thus, multiple MJoin operators are used for the establishment of a global shared query execution plan and to mitigate the window update and rooting problems associated with join operation results. Moreover, an efficient multi-query optimization and processing technique is used for the stream data environment of IoT [12]. This study also evaluates the system performance through experimentation according to the changes in the sliding window size as well as the optimal (sigmoid) kernel function of the SVM classification algorithm for efficient storage management. Finally, to implement a system that can actively respond to users, classified and reduced sensor data were utilized to enable intelligent control of devices inside the home [13].

II. RELATED RESEARCH

IoT environments are composed of three technologies: sensing technology, which measures changes in the environment; interface technology, which performs or links certain features through people, things, and services; and network infrastructure technology, which creates networks between sensors and services [14]–[16].

These technologies provide a variety of environments, including remote control, which operates according to the user's needs, and automatic control, which recognizes people and provides custom services [17], [18]. This chapter analyzes conventional systems that use automatic and remote control to create intelligent smart home systems, as well as the requirements for IoT.

A. IoT AUTOMATIC CONTROL SYSTEM

Automatic control uses control devices, machines, and computers to automatically perform control operations [19]. Tasks are configured automatically without user intervention. Automatic control services can increase the efficiency of device usage in an environment using several IoT devices simultaneously, and the importance of these services is becoming clear [20].

Automatic control systems are used in many fields, including smart homes, which provide customized environments via devices in the home; self-driving cars, which drive by themselves without operation by a human; and healthcare for user health management and disease prevention.

B. IoT REMOTE CONTROL SYSTEM

Remote control involves the indirect control of a device without a person directly operating the device using a communications network. Currently, as the demand for remote control is increasing, IoT has evolved into an environment that can easily be controlled by the user without any knowledge of IoT.

Remote control systems are used to remotely read the usage of gas, water, and electrical energy in a household. They are also used in industrial sites and smart buildings, which require continuous management. In addition, remote control can use the user's smart devices to operate devices within a home or perform tasks as required by the user.

In cloud-based IoT framework structures, the app layer is divided into the IoT sub and non-IoT sublayers. In the IoT sublayer, structured and unstructured data are created by sensors and devices. In the non-IoT sublayer, the user's remote requests are transmitted to the cloud.

The cloud processes these requests in a handler and transmits them to a manager to share them with other apps. In this manner, the processing time in large-scale systems can be reduced by skipping the processing step when user requests stored in the cloud are used. This approach also provides custom services that can remotely control devices in systems that have similar IoT environments [21].

C. ANALYSIS OF THE REQUIREMENTS OF IoT SYSTEM

The systems developed thus far to control IoT-based environments are able to measure the sensor data and statuses of devices to automatically create environments that are pleasant without user intervention.

In addition, they can remotely check and control the status of an environment in real time [22]. These systems measure the statuses of devices and sensor data and transmit these to the server. Because the server processes these events in batches, this has the drawback of processing high-priority events after finishing the events that are currently being processed [23], [24].

Context awareness/monitoring and remote control are needed to satisfy the requirements of IoT systems [25]. These requirements are as follows.

1) CONVENIENCE

Settings and selections by the user must be minimized, so that the user can simply experience the provided service.

2) ACCURACY

System operations must not diverge from the user's intentions, and there must be no malfunctions owing to system problems.

3) OPERATIONAL CHANGES

Device operations must change, and the system must provide services that operate according to changes in the surrounding environment (classified by the usage date and time) and the home's internal status (order of priority).

4) USER INTERVENTION

The user must be free and able to intervene in the system at the desired time during operations.

III. SYSTEM CONFIGURATION AND DESIGN

Join queries are required to process comprehensive data that are acquired from not just one sensor, but multiple sensors, as in the case of an IoT environment. Previous studies on stream data have proposed techniques for establishing query execution plans for join queries with regards to the cost and storage space efficiency. Join operators include operators based on hash tables, windows, and both hash tables and windows [26], [27].

Of these, hash table-window join operators are the most suitable processing method for high-capacity data streams, because they can operate with limited memory and provide a quick matching speed. The MJoin operator was proposed as a hash table-window join operator that can select multiple inputs, in view of the fact that the results of joining several data include more comprehensive content [28]. This study used the hash table-window operator multiple MJoin to optimize queries.

In addition, the system was implemented such that it could provide an intelligent environment by activating task events according to the classified and reduced sensor data and setting the priorities of operations with a focus on the user. Conventional home systems transmit commands to objects that are connected to the same network if a threshold value related to the task is reached. The objects execute the tasks, but the system does not consider multiple tasks that occur simultaneously [29].

As such, this has the disadvantage of executing unnecessary tasks and wasting electricity. In this manner, systems in an IoT environment must process not just singular data, but multidimensional data simultaneously. Therefore, they must process stream data more quickly and set priorities among tasks such that high-priority tasks are processed.

The IoT-based smart home system transmits data that have been measured by the sensors to the server and uses the multiple MJoin operator to optimize queries.

Furthermore, it uses the SVM classification algorithm to classify and reduce data, and the classified and reduced data are saved in the TinyDB. Real-time and non-real-time tasks are differentiated in the server. Real-time tasks are provided to the user, and then saved in the MainDB. Non-real-time data are saved in the MainDB and provided if requested by the administrator.

In addition, the priorities among tasks are set such that the number of simultaneous tasks can be reduced by suspending existing tasks when a higher-priority task occurs, to reduce unnecessary and wasteful electricity usage while providing intelligent services to the user.

Figure 1 illustrates the proposed system's configuration.

A. SENSOR PROCESSING AND DATA STRUCTURE

To process the sensor data of the IoT-based smart home system, the input stream data are collected from each sensor. After scanning the TinyDB, preliminary clustering is performed. When the scan is complete, the stream data are saved. Multiple MJoin query is performed on the data, and it is saved. Then, the stored data are classified by the SVM classification algorithm, and the data are reduced.

An Arduino (Uno) module is used as the processor board, and five sensors are used to acquire streams (temperature, humidity, gas, vibration, and detection) for sensor data processing. The data to be used in the analysis all originate from the same environment. Therefore, they are combined into one packet and transmitted. Each packet generates additional traffic and consumes energy. Therefore, a single packet is used to process a query [30].

The packet's total length is 36 bytes, including 10 bytes for the fixed header, 6 bytes for the sensor node ID and channel, and 20 bytes for the buffer. Of these values, the first 12 bytes of the buffer are designed to contain the actual sensing values in 2-byte units in the order of temperature, humidity, gas, vibration, and detection.

To smoothly control the sensor flow, the system requires the sensor data generated by the sensors and user task commands. The system checks the frequency and time of the sensor data and generates the results data. The results data are classified according to which of the various tasks a sensor corresponds to and are saved in the database according to the date and time.

As a basis for checking the month, week, and day, it must be possible to analyze the sensors' usage periods by month, week, and day, because they vary slightly according to the various environments and situations.

Based on the usage rates of the sensors analyzed by month, week, and day, sensors with low usage rates are changed to a standby or drop state. When the sensors are in the standby state, they do not generate data, which reduces data waste.

Furthermore, in the drop state the user directly turns the sensor's power off, which can reduce electricity waste. The user can control each sensor. The data occurring in the sensors of the TinyDB database are saved in the database. The system refers to this database to perform AP tasks.

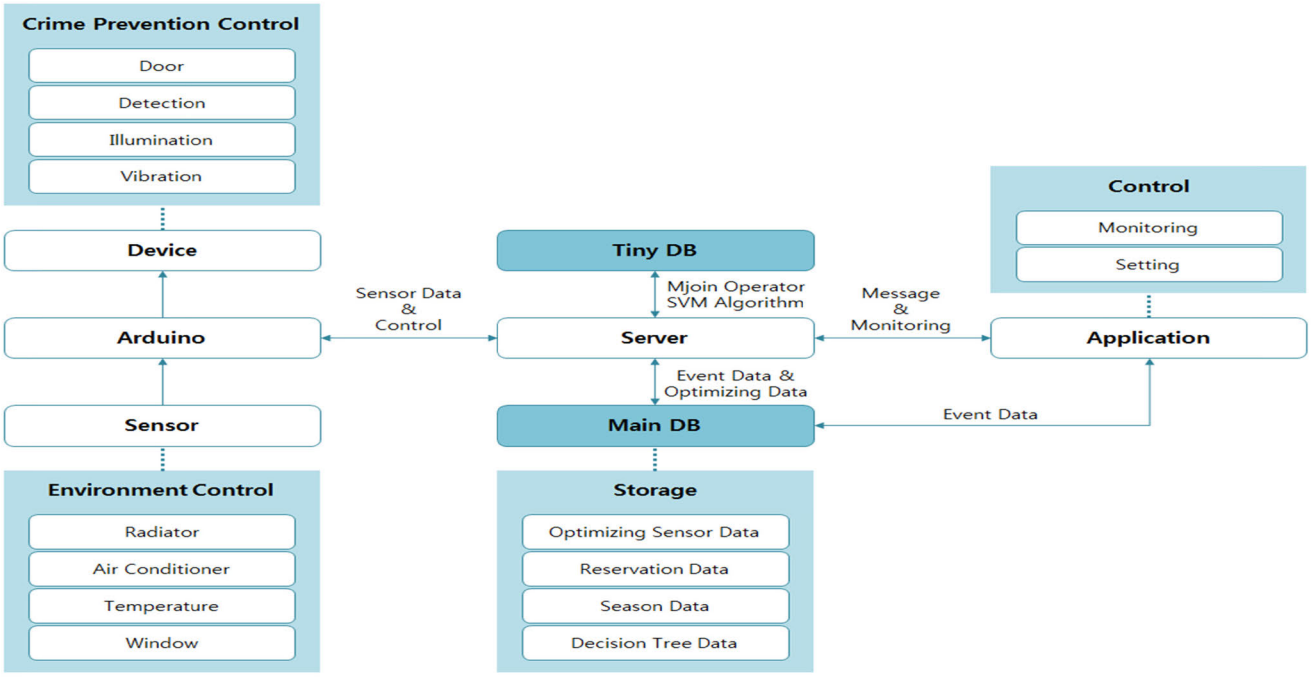


FIGURE 1. System configuration.

Users use applications to set the sensor threshold value range, and a device can be controlled by checking the sensor data. The range values set in the database are loaded and referred to when performing tasks.

The sensors consist of temperature, humidity, gas, detection, and vibration sensors in the home. The temperature sensors were designed to be combined with the gas sensors such that they can handle dangerous situations. In addition, tasks performed with several sensors are digitized and stored in the database. Based on the sensor data, the system provides a convenient IoT environment by performing tasks in which the events that are related to the tasks are suitable for the situation or environment. Based on the sensor data range set by the user and the database data, the user analyzes the data in the application to control the sensors. Sensors with low usage can be placed in a standby state, and a sensor's state can be changed to be operational again.

B. WINDOW DEFINITION AND STREAM PATTERN CONDITION PROCESSING

Window processing of stream data refers to the division of incoming data into window units in real time. Here, the interval and size used to define a window for the window operations are the values saved in a database by a user. The window interval is the range over which the window moves by a specified distance over the data stream, and the ''size'' is the unit in which data are aggregated.

After window processing, the data segments divided by windows are processed according to the pattern condition. ''Pattern condition'' refers to the classification method for generating a training data set by the user. The stream patterns in this study are divided into five: a filtering pattern for removing outliers, an increase or decrease pattern, a pattern for detecting values above or below a threshold, a processing pattern applied after calculating the sum and average of the data in a window, and a pattern to be compared with the user-designated pattern.

To recognize all user-defined patterns, these five divisions (or parts) were classified considering the user's understanding and the computational complexity involved in the process. Because any part can be defined by a user when a pattern is executed, each part is classified based on the most universal and easy to understand criteria to facilitate system use. Additionally, quick processing is required to classify incoming data in real time. Therefore, parts are separated by complexity to prevent long computing time and efficiently classify stream data.

Table 1 shows an overview of the definitions for the five classified patterns.

1) FILTERING PATTERNS FOR ELIMINATING OUTLIERS

The pattern for removing outliers is used to remove incorrect data. When data are input from a sensor, the value of the data is influenced by the environment. Thus, the likelihood of incorrect data being generated increases if the device operates in a harsh environment or in a remote location. In addition, bad data can be generated from sensor malfunctions. Bad data contained in training data can impact the accuracy of training results. Therefore, the user must store threshold values in the database so that values above or below the threshold are

TABLE 1. Stream pattern definition.

No	Classification	Definition
1	Filtering patterns for eliminating outliers	A pretreatment process that removes data other than user-defined intervals to extract only valid data values
2	Increasing or decreasing pattern	Continuous increase/decrease, increase/decrease compared to the first and end values in the window, increase/decrease considering limited variability
3	Above and below value detection pattern	Pattern considering the number of data values above and below the user defined threshold
4	A pattern for processing the sum and average in the window after calculation	Comparison with user set value after sum / average calculation process in window
5	A pattern to be compared with a user designated pattern	Pattern detected by match / intersection / difference of sets/ correlation analysis with user specified pattern

excluded before generating the training data set. The system in this study allows the user to decide whether or not to filter outliers.

When filtering is performed, the resulting data (without outliers) are used for processing other pattern conditions and for generating the training data set. When filtering is not performed, the training data set is created using all the generated data.

2) INCREASING OR DECREASING PATTERN

The increase or decrease pattern condition extracts a window in which the data values tend to increase or decrease. An increasing pattern data set can be used for detecting increases in fine dust concentration or sudden increases in blood pressure.

Accordingly, this study defined three detailed conditions for generation of the increasing training data set: a series of values in a window constantly increases, an increase occurs based on a comparison of the first value and end value in a window, and an increasing trend is shown overall even if the values in the middle of the window decreased. For the third condition, two sub-conditions must be met: first, an increase must have occurred based on a comparison between the first and last values (i.e., the second condition above). Second, the user stores a deviation value and its frequency in the database, and the degree of decrease is in a value falling within the stored deviation.

Figure 2 shows three example windows representing the increase conditions.

Similar to increasing pattern, the decreasing pattern was also divided into three conditions. Continuous decrease was divided into two cases: a decrease based on a comparison of the first value and end value in a divided window and

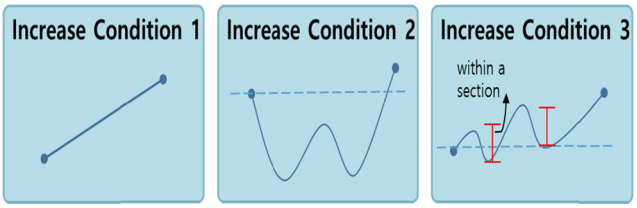


FIGURE 2. Example of window data by increasing condition.

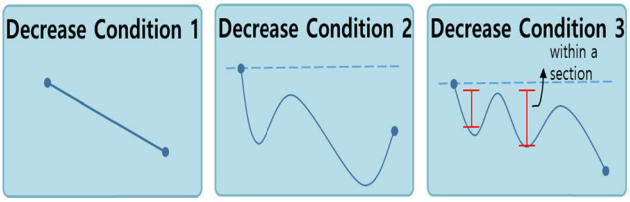


FIGURE 3. Example of window data by decreasing condition.

an overall decreasing trend even if the values in the middle increased. For the third decrease condition, opposite to the third increase condition, a decrease must have occurred based on a comparison of the first value and end value, and the degree of increase is a value within the interval and its frequency allowed by the user. The decreasing pattern can be used to extract data sets that only exhibit decreasing trends, such as decreasing body temperature or atmospheric oxygen concentration.

Figure 3 shows examples of the decrease condition.

3) ABOVE AND BELOW VALUE DETECTION PATTERN

In the pattern for detecting values above or below a threshold, if a value above the threshold is detected, then a window with a value larger than the user-specified value among the data values in the window is extracted. Further, if a value below the threshold is detected, then a window with a value lower than the stored value is extracted. Here, the frequency of values above or below the threshold in the window is also stored in the database, and the window is extracted to evaluate a data stream that exceeds the stored frequency

4) A PATTERN FOR PROCESSING THE SUM AND AVERAGE IN THE WINDOW AFTER CALCULATION

When operation of the values in a window is needed, the sum and average of the values are calculated to determine the pattern to use, after which the window corresponding to the condition stored by the user is extracted. The user stores a number in the database that is used as the criterion as well as the condition for comparison (greater than, less than, greater than or equal to, less than or equal to, or equal to). Only the window whose calculated sum or average value meets the comparison condition with the criterion value is extracted. In this case, it is compared with the pattern specified by the user and is then extracted.

5) A PATTERN TO BE COMPARED WITH A USER DESIGNATED PATTERN

The user first specifies a pattern in the database and then the window that meets the classified criterion is extracted. There are four different conditions that are used for comparisons.

The first condition is to extract a window that matches the stored pattern, which can be used to obtain information on how many times a match with a given pattern occurs within a given time.

The second condition is to extract a window in which an intersection value exists. Here, a window is extracted if it has a value that intersects with a pattern; this applies if the stored pattern value exists regardless of the time sequence. This condition can be used to track the flow of values generated in a certain pattern.

The third condition is to find a window containing value(s) other that of the stored pattern, which can be used to find windows indicating dangerous or unusual values.

The last condition is for conducting a correlation analysis. A correlation analysis is conducted between the stored pattern and the values of an incoming window, and the result is calculated. Based on the correlation analysis result, such as a total or average operation pattern in the window, a window is extracted if the result meets the condition for the value stored by the user (based on a greater than, less than, greater than or equal to, less than or equal to, or equal to comparison).

C. MULTIPLE QUERY PROCESSING WITH MULTIPLE MJOIN

To obtain comprehensive data from the sensor network, the join operation must be performed based on a specific time or location, and its results must be obtained. In this study, a query plan was established using the MJoin hash table-window join method. MJoin is a method for efficiently joining data streams that frequently change. It is an extension of the symmetrical hash algorithm, such that the processes of multiple streams are available. It repeatedly checks for the existence of tuples that have the same key in all hash tables for each input tuple.

Normal binary join-based join queries have a problem with blocking, because they establish a query execution plan in the form of a binary tree. In a data stream environment, a potentially unlimited amount of data is continuously inputted into the system, and therefore a query execution plan that experiences blocking will exceed the system's memory limitations and require input stream sampling or load shedding.

MJoin has been proposed as an efficient join processing technique for multidimensional stream data that diverge from this kind of join-based form and can accept multiple streams as input. MJoin was developed using the traditional symmetric hash join. In other words, unlike the conventional symmetric hash join, MJoin can accept multiple inputs.

Therefore, it does not pass the intermediate results to the next operator and produces the join results for multiple streams.

Figure 4 illustrates the MJoin processing structure.

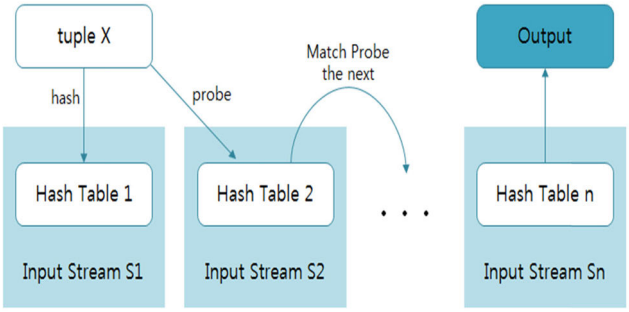


FIGURE 4. Processing structure of MJoin.

If a new tuple arrives from the input stream S1, then this tuple is inserted into the hash table for S1, and the hash table for the next input stream is examined. If the newly inputted tuple matches all the values in the other hash table, then the results are produced.

In this study, after identifying the containment relationships among MJoin operators, a global shared query execution plan was established.

The multiple MJoin technique optimizes multiple MJoin operators considering multiple-input support provided only by MJoin operators, an optimized evaluation sequence, and tuple usage to implement a sliding window in a data stream environment. In addition, the processing result of the join operator can be reused as input for the parent join operator. In this environment, an index-based purging tuple that solves the window update problem for the join operation result is used. Finally, to solve the routing problem of the resulting join operation tuple (hereafter referred to as the join operation tuple), the dead vector technique, and dead tuples that propagate newly added dead values to the parent join operator are used. Similar to purging tuples, dead tuples are used when performing an index-based search to identify which windows are not satisfied by the join operation tuple.

The problem of establishing an optimal global shared query execution plan for multiple MJoin operators is NP-hard. Furthermore, owing to the data stream environment of IoT, an approximation technique that can approach the optimal solution at high computational speed is required, and the following conditions must be satisfied:

- Satisfactory performance must be achieved, even with a reduced search space and few searches conducted.
- The characteristics of the MJoin operators must be sufficiently reflected.
- A sliding window constraint that allows only limited memory to be used in a data stream environment must be reflected.

First, owing to the characteristics of MJoin rather than to the set of all possible query execution plans, the search space can be greatly reduced by searching based on a single query. Second, excessively splitting the operators can result in performance degradation. Increasing the number of operators increases contextual exchange and tends to violate the optimized probing sequence. It is instead preferable to establish

a global shared query execution plan so that the number of returned operators is equal to or less than the number of independently executed MJoin operators. Third, the scale of performance improvement increases as more operators with high processing costs are shared. That is, rather than processing multiple high-cost operators separately, it is more beneficial to process them simultaneously. With these requirements in mind, the cost model of the joint operator in this paper is defined as follows.

Suppose that a given MJoin operator uses n values of input $R = \{R_1, R_2, \ldots R_n\}$, where the input speed and window size of each R_1 is r_i, W_i, and the selectivity factor of the join operation is f. With these parameters, the cost model can be defined as in Equation 1.

$$C_{R_1,R_2,R_3\ldots R_n} = \prod_{\substack{all \\ selectivities}} f \cdot \sum_{k=1}^{n} \left(r_k \cdot \prod_{\substack{i=1 \\ i \neq k}}^{n} W_i \right) \tag{1}$$

In addition, the window size for each join operation result can be predicted using Equation 2. This equation can be used to measure the cost of the parent join operator.

$$C_{parent} = \prod_{all} f \cdot \prod_{i=1}^{n} W_i \tag{2}$$

Finally, the sliding window constraint must be taken into account. The join operators can be shared and processed simultaneously, even if the windows defined in each query differ. When an operator is shared, it will contain the largest window size, meaning that it can store more tuples after the operator is shared than if being processed as an independent operator.

Suppose that there are m shareable queries $Q_i (1 < i < m)$, the shared MJoin operator has k input streams $R_j (1 < j < k)$, each query Q_i uses the window size W_{ij} for each input R_j, and the largest window size defined for all queries for each R_j is W_j^*. With these parameters, the processing amount is reduced when the operations are shared by meeting the condition stated by Equation 3.

$$\sum_{i=1}^{m} \prod_{j=1}^{k} W_{ij} \geq \sum_{j=1}^{k} W_j^* \tag{3}$$

Based on the above, the multiple MJoin optimization algorithm that approximates and establishes the optimal global query execution plan of multiple MJoin operators is implemented as shown in Figure 5. Each query is represented by one operator, and each operator can be represented by a set of input streams.

As indicated in the multiple MJoin optimization algorithm, one query, that is, one MJoin, is selected at every step. In each step, if a plurality of the most shared sets is selected, then the set with the highest processing cost among these is selected. If the selected set does not meet the sharing condition, then

```
Input : a set of queries, QuerySet[]
Output : shared query plans, QuerySet[]
while QureyCount > 0
begin
    for i :=0 to QueryCount
    begin
        if Containing # of SelectedSet < Containg # of QuerySet[i] then
            selectedSet :=QuerySet[i]
        else if Containing # of SelectedSet = Contain # of QuerySet[i] and
                Cost of QuerySet[i] > Cost of SelectedSet then
            selectedSet :=QuerySet[i]
    end
    if SelectedSet satisfies CONDITION OF SHARING then
    begin
        for i :=0 to QueryCount
        begin
            Replace the elements of QuerySet[i] to common elements SelectedSet
        end
        for i :=0 to QueryCount
        begin
            if QuerySet[i] has only one element then
            begin
                exclude QuerySet[i]
                Querycount : = QueryCount -1
            end
        end
    end
    else
        exclude SelectedSet
end
return QuerySet
```

FIGURE 5. The multiple MJoin optimization algorithm.

the algorithm excludes the selected set and proceeds to the next step. If the selected set meets the sharing condition, then the part containing the elements of the selected set is searched for within all given sets (including the selected set), and that part is replaced with the selected set. Thus, all common elements are replaced by one element. Finally, sets with only one element are excluded from the next step because a set having one element signifies that the query execution plan has already been completed. These tasks are repeated until all queries are excluded.

Each time a tuple is removed or routing information is added by a window update in a raw data stream, all join operation tuples related to this tuple must be reflected to correctly perform a window update and routing. To accomplish this, an index assignment scheme is used. That is, by assigning an index, it is possible to identify from which tuples each tuple is generated or, conversely, which tuples each tuple generated. This requires that two types of index information, PrevID and PostID, are written in each tuple.

In addition to assigning an index, routing information is added to display more detailed location information for each tuple. Routing can be considered as a task that assesses which queries are satisfied and which are not satisfied by the join operation tuples based on the window defined for each input in each query when the join operator is shared. Therefore, branch information must be calculated before the join operation tuple is sent outside the operator.

The dead vector technique was originally proposed for the shared processing of join operations and select operations. When executing a select operation, the join operation is executed while recording the match information in the dead vector of the tuple without removing the tuple. The original dead vector is merged into the generated tuple and routing is executed using that information. In this study, the dead

vector is used to write the routing information in the tuple in advance, and after the join operation, the information is used to branch to each query.

Each time a tuple is removed from a raw stream tuple or a new dead vector is added to a tuple, the result tuple of the related parent join operator must also be affected. This study used the generation of purging tuples and dead tuples to carry out this task. These two tuples are used to apply a variation of the negative tuple technique; they differ in that the negative tuple technique executes a value-based search, whereas the purging tuple and dead tuple execute an index-based search. Another difference is that the join operation is not repeated to generate the purging tuple and dead tuple.

■ Generation of Purging Tuples

Each tuple keeps information regarding which tuples were generated by and which tuples were generated based on the index values. Using this information, when a tuple is removed during window updates, the tuple(s) associated with it should also be deleted. The purging tuple is created for this purpose. The purging tuple, which is used only to delete tuples and does not participate in actual join operations, has index information, and a special flag indicating that the current tuple is a virtual tuple.

■ Generation of Dead Tuple

If a new value is added to the dead vector of a raw stream tuple, then the same value must be added to the dead vector of the join operation tuple generated from this tuple. This study proposes the dead tuple as a tuple that carries this information and can add values to the dead vector. If the dead tuple is transferred to the parent join operator, the PostID value of the dead tuple is obtained, the index hash table is used to find the tuples with the same PrevID in the join context table, and the dead vector with the dead tuple is combined with the matching tuples.

The purging tuples and dead tuples perform an index-based search to remove these tuples or add new dead vector values. If the PrevID of a tuple is included in the PostID possessed by a purging tuple or dead tuple, then the tuple is removed or a new dead vector value is added. If this search is executed without a separate data structure, then all tuples in the join operator are evaluated based on the index. In this study, an index hash table is built within the join operator to more quickly perform index-based searches. The index hash table is constructed based on the PrevID of tuples in the corresponding join operator, which enables quick searching for the desired PrevID when purging tuples and dead tuples are used.

When purging tuples and dead tuples are generated and sent to the parent join operator, the PostID of the purging tuple and dead tuple is read in the parent join operator, and the tuples corresponding to this value must be removed or a new dead vector must be added.

The validation algorithm for join operations presented in Figure 6 is a tuple validation algorithm using the index assignment, dead vector, purging tuple, and dead tuple techniques discussed thus far.

```
Input : a input tuple, tuplenew
if tuplenew is Purging Tuple then
begin
    for each id ∈ PostID values in tuplenew do
    begin
        remove a tuple with PrevID value = id from correlate State
        generate Purging Tuple tuplepurging for a removed tuple
    end
end
else if tuplenew is Dead Tuple then
begin
    for each id ∈ Dead values in tuplenew do
        begin
            for each id ∈ PostID values in tuplenew do
            begin
                add d in a tuple with PrevID values = id as Dead Value
                generate Dade Tuple tupledead for the tuple
            end
        end
end
begin
    for each w ∈ Window sizes in join operator do
    begin
        if oldset tuple in state is out of w
            add Dead in the tuple for w and generate Dead Tuple tupledead
    end
    if oldset tuple in state is out of maximum window size
        generate Purging Tuple tuplepurging
end
```

FIGURE 6. The validation algorithm for join operations.

D. SVM CLASSIFICATION ALGORITHM

The data are classified and reduced to efficiently manage the stream data storage while taking the IoT environment into account. Various learning algorithms exist, but the data used in this study are temperature, humidity, gas, vibration, and detection data, and the structure of these data is nonlinear.

As such, this study used the SVM classification algorithm, which has a multilayer perceptron structure and can solve nonlinear discriminant problems.

The SVM classification algorithm, which is based on learning theory, finds the solution by always converting the problem into a convex quadratic problem, which ensures a globally optimal solution. Because of this, it yields an excellent performance in the field of pattern recognition [31].

The algorithm used in this study is a dual SVM classification algorithm, and it classifies whether the given data are in a certain range. The data in the range are saved in the database, and those that are not are automatically deleted, to increase the efficiency of the database. The SVM classification algorithm is a method that finds a classification hyperplane that suitably separates the two groups [32].

The SVM classification algorithm has better scalability than conventional linear classification algorithms, and always yields an excellent performance, unlike neural network classification methods, which produce different performances each time they are trained [33].

The basic principle of SVM starts with linearly separable problems. Given the input data X_i in dimension d, the problem of classifying the training data into binary out values such as -1 and $+1$ is considered. To define a model for classifying the two groups, a hyperplane can be defined as a linear discriminant function. Here, a support vector indicates a sample that has a close relationship with the boundary that determines the classification rule.

In the case that data are not linearly separable, such as sensor data, the nonlinear mapping $\emptyset$ is used to convert the data to

TABLE 2. SVM kernel representative types.

Algorithm : SVM Number of Data for Learning : N Inputs: Sample x to classify Dataset : I_i $I_{i1} : temperature, I_{i2} : humidity, I_{i3} : gas, I_{i4} : vibration\ I_{i5} : sensing$ Output decision u $\in -1, 1$ Classify using SVM get the result in the form of a real number

TABLE 3. Kernel representative types of SVM classification algorithm.

Type of SVM	Kernel Function Expression	Parameter
Gaussian or Radial Basis Function (RBF)	$\mathrm{K}(x_i, x_j) = e^{-\left\|x_i - x_j\right\|^{2/2\sigma^2}}$	σ
Linear	$\mathrm{K}(x_i, x_j) = x_i^T \cdot x_j$	-
Polynomial	$\mathrm{K}(x_i, x_j) = (x_i \cdot x_j + 1)^d$	d
Sigmoid	$\mathrm{K}(x_i, x_j) = \tanh(\gamma x_i \cdot x_j + C)$	γ, C

a dimension that is higher than the input vector's dimension and can be linearly classified. Then, linear classification is performed.

In the nonlinear mapping, a kernel function is used to convert the N-dimensional input space data to a high-dimensional feature space (Q dimensions) so that it can be linearly classified. Equation 4 shows the kernel function and determination function.

$$\begin{aligned} \mathrm{K}(\mathrm{x}, \mathrm{y}) &= \emptyset(x) \cdot \emptyset(y) \\ \mathrm{f}(\mathrm{x}) &= \sum_{i=1}^{n} a_i y_i K(x, x_i) + b \end{aligned} \tag{4}$$

The SVM classification algorithm does not perform tasks such as finding the classification hyperplane and minimizing the sampling error. Rather, it increases the classification accuracy for new data by maximizing the classification margin. Because the SVM classification algorithm was developed for binary classification, it faces considerable difficulty in solving problems that include several classes in an actual environment.

To solve these problems, the one-against-all and one-against-one techniques have been proposed. The one-against-one technique consists of k(k-1)/2 SVMs, where k is the number of input classes.

The training speed is fast, because the training data consist only of data that have two affiliations, and the amount of training data used for each training session is small. To improve the training performance, the one-against-one technique was used to perform experiments, and the SVM classification algorithm was configured as shown in Table 2.

The SVM classification algorithm creates a hyperplane that generates many maximum margins for each feature of the given training data. In the test stage, mapping is performed on a multidimensional space that is divided by the hyperplane created in the training stage, and new data are classified.

The representative kernel functions of the SVM classification algorithm used to train the SVM classification algorithm are listed in Table 3, where γ, C, and d are the parameters that determine the form of the kernel function. In the SVM classification algorithm, it is important to select both the appropriate kernel function for the given task and various parameters in the SVM kernel function. Both of these processes have a decisive impact on the efficiency and accuracy of SVM classification. A grid search (GS) algorithm was used for parameter optimization. GS tests discrete values at reasonable intervals within a given range to find the optimal parameters. The optimal kernel function is the one that achieves the highest accuracy. In addition, a prediction of how well the model performs classifications is necessary. To objectively measure the final performance, this study used the polynomial, radial basis function (RBF), and sigmoid kernel functions under tenfold cross validation.

E. IoT-BASED SMART HOME SYSTEM

The IoT-based smart home system distinguishes the sensor data that have been classified and reduced by the SVM classification algorithm and sets the priorities among tasks. This was designed to operate tasks and devices according to the environmental changes within the home. It was also designed such that the classified and reduced sensor data can be transmitted to the server, and events can be processed using threshold values.

The in-home status can be monitored by applications, and administrators can remotely control devices or receive messages when dangerous situations occur. In this manner, the system can provide an intelligent service with enhanced convenience, because it reduces the manual work performed by the user in response to changes in the in-home environment.

The smart home system transmits the sensor data to the Arduino (Uno) module. The transmitted sensor data are classified and reduced after the query is processed, and then saved in the TinyDB.

In the server, the system determines whether the sensor data are being used by extracting the sensor data and environment information (device data and date), and the system recognizes the situation. Tasks are set according to the recognized situation, and the priorities are set according to the values selected by the equipment and provided to the devices.

The MainDB stores the classified and reduced sensor data and device statuses following real-time or non-real-time processing, the usage date and time data, the member data for distinguishing administrators, and the administrator tokens for transmitting notification messages during dangerous situations. The application can be used by the administrator to monitor the current status within the house and remotely control devices when needed.

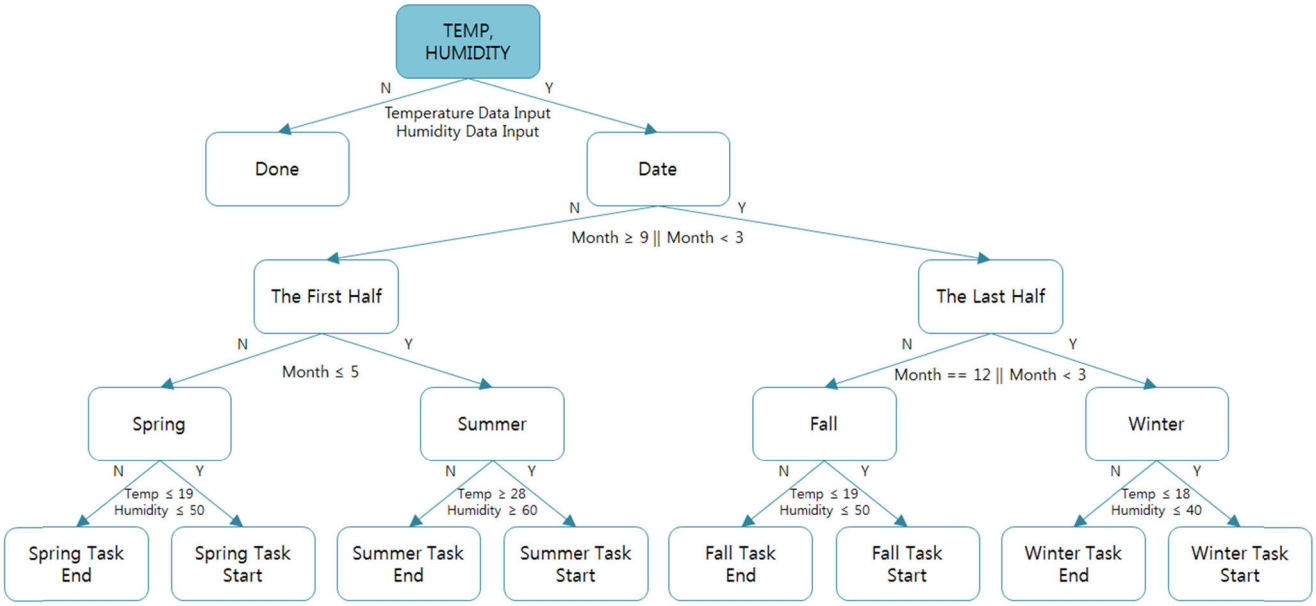


FIGURE 7. Temperature & humidity decision-making tree.

The sensor data measured to recognize the smart home system's status (temperature, humidity, and gas) and the date data were classified into each task using a decision-making tree. The date data were classified into seasons and tasks, and the sensor data were used to set the devices.

Figure 7 shows the proposed decision-making tree for sensor data (temperature and humidity).

The system determines whether the temperature sensor values have been inputted and begins classification. If the temperature sensor values have been inputted, then classification begins, and the current date is used to classify times into the first and second halves of the year. If the temperature sensor value has not been inputted, then classification does not begin, and the standby state is maintained. If classification into the first and second halves of the year has been completed, then the current month is set as the classification standard for determining the season.

After the season has been classified, the current temperature value is compared with the seasonal threshold value, and the task operation is determined. If the task has begun, then a device that will operate during the season's task is selected and transmitted to the automatic relationship-setting module.

Like the temperature sensor, the system also determines whether the humidity sensor value has been inputted and begins classification. After the system completes classification into the first and second halves of the year and into seasons, the humidity sensor value is compared with the seasonal threshold value. The task operation is determined, and the devices are selected.

Figure 8 shows the decision-making tree for the gas sensor data. When the gas values measured by the sensors have been inputted, it is determined whether the threshold value has been exceeded and a gas leak has been detected. When a gas leak is detected, the gas leak warning message implemented by Firebase Cloud Message (FCM) is sent to the user. After the gas leak is detected, the temperature sensor's value is received as input. If the temperature sensor value exceeds the threshold value, then a fire is detected, and a fire alert message is sent to the user.

In this manner, the system determines whether sensor data (temperature, humidity, and gas) have been inputted, and begins classification. In the decision-making tree of each sensor, the season is classified by the data and sensor values, and the task is classified according to the season.

The reason that the season is classified using a decision-making tree is that the selected devices and device operations must change according to the environmental changes within the home, and the types of active devices and tasks are different for each season. The task operations are determined to set the device and its status by comparing the classified task's threshold value with the current sensor value. By setting threshold values for each season, the environment within the home can be maintained as the seasons change

In addition, five sensors are used to intelligently control the priorities for ventilation, temperature, fire, and intrusion. The relationships between the devices are set to prevent errors and bottlenecks resulting from operating the selected devices simultaneously.

The statuses of the devices selected via device classification are updated in the device database, and then the devices' statuses are compared in the relationship-setting module. Simultaneous operation of the devices can be prevented by setting their priorities and operating them sequentially [34].

Figure 9 shows the priorities for the tasks.

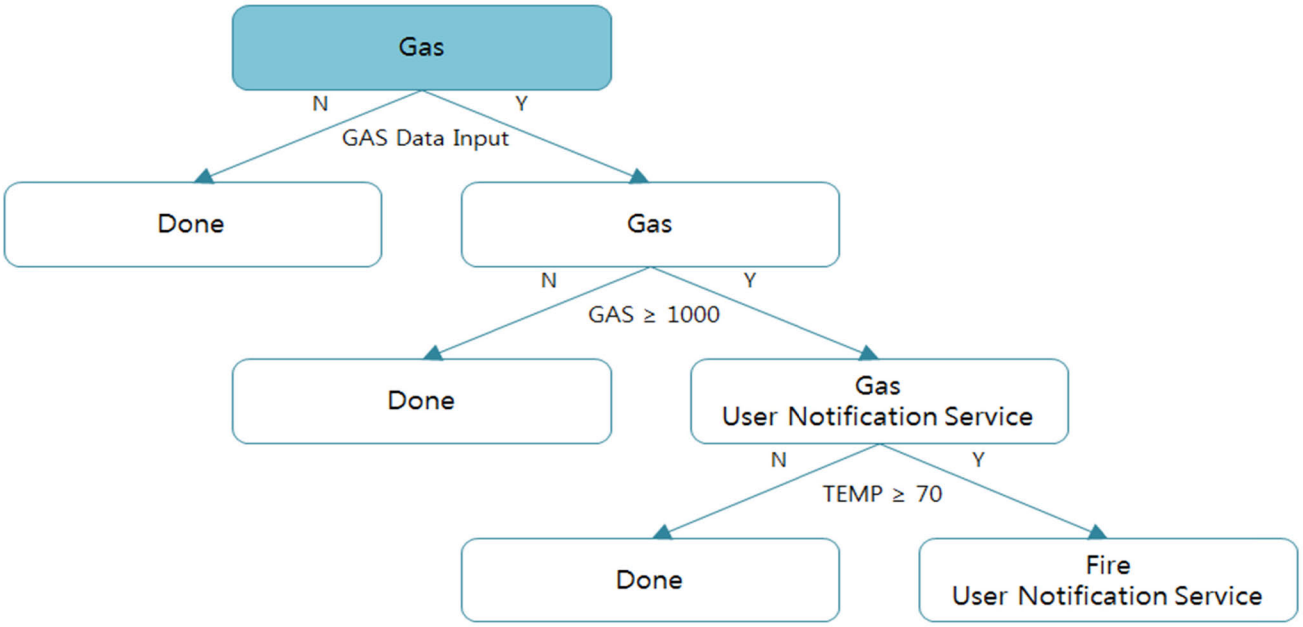


FIGURE 8. GAS decision-making.

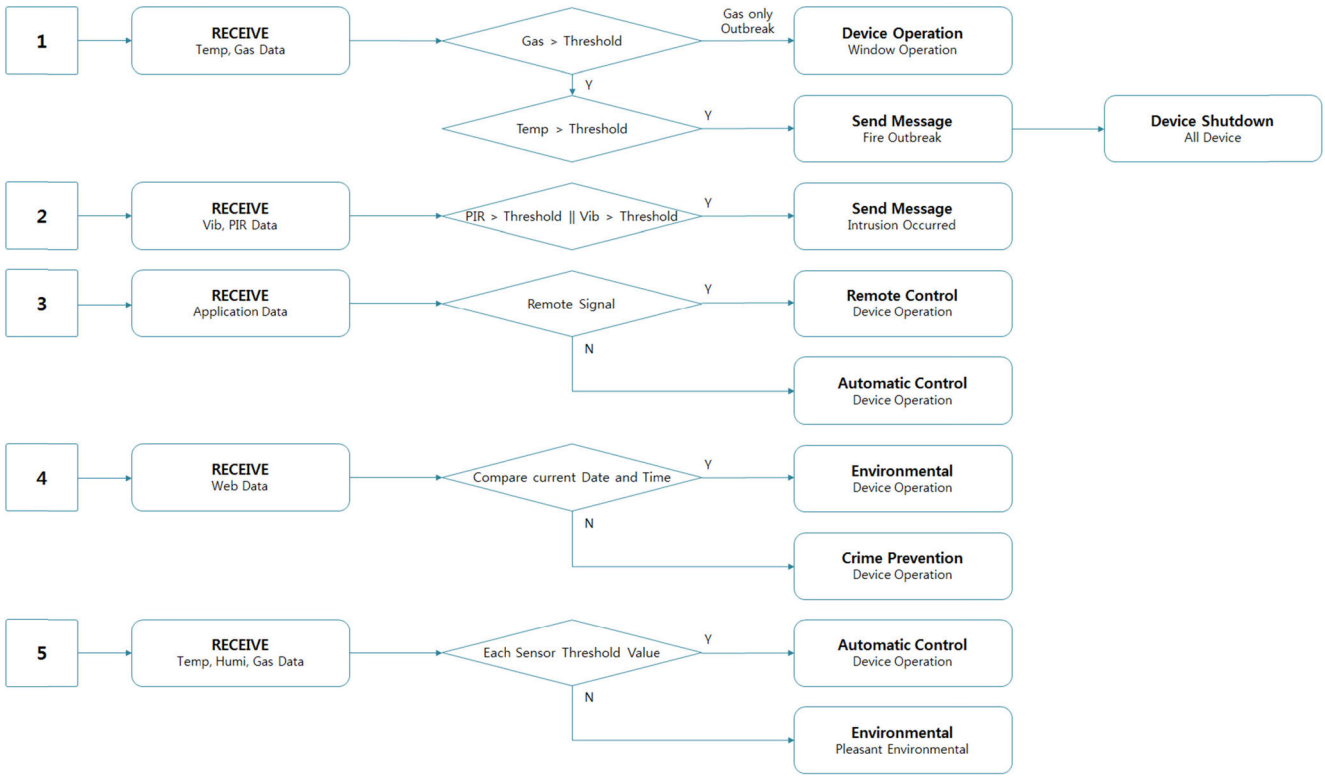


FIGURE 9. The priorities for the tasks.

The smart home system was designed such that an application could be used to monitor and control the statuses of devices, as shown in Figure 10. When the user changes a device status, the selected device's status data are transmitted to the server and saved in the Devices table. For the saved data, a device control command is transmitted to the Arduino (Uno) module via the Wi-Fi shield [35].

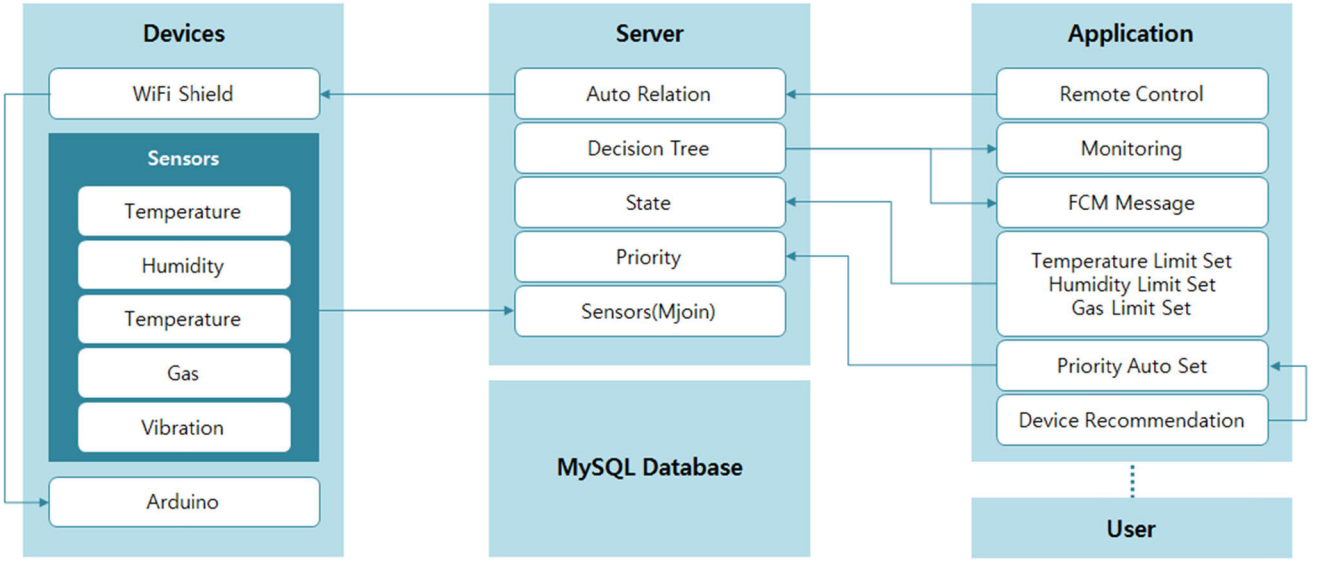


FIGURE 10. Smart home system using application.

The Arduino (Uno) module executes the command according to the change in the device's status. If the user wants to include the device's operation in the task again and execute it, then the device's status is changed to standby via the application's remote control, and the operation can be performed through the connected task. As such, the user can intervene at the desired moment and control the device's operation, and therefore user-customized services can be provided [36], [37].

IV. EXPERIMENT AND IMPLEMENTATION

To evaluate the performance of the system proposed in this study, a control environment was created from an Arduino (Uno) module, a Wi-Fi shield, and five sensors (temperature, humidity, gas, vibration, and detection). The system employed Windows as the operating system.

The CPU was an Intel i5-4460, and the system had 8 GB of RAM. A MySQL database was used for both the sensor data and Arduino remote control data. An Advance Power Management (APM) Setup was used to communicate the measured data.

A. MULTIPLE MJION QUERY PROCESSING OPTIMIZATION

In the multiple MJoin technique, by sharing operators, a join tree is formed, and purging and dead tuples are used for window updating and routing of the join operation results in the tree.

In this experiment, we assumed 20 input streams using various environmental sensors for the optimization of multiple Mjoin query processing. Given that various sensor data are generated without noticeable delay in an Internet-of-things (IoT) environment, we conducted experiments on the optimization of the multiple Mjoin query processing. This experiment involved 20 types of information flow into a single system. There are 20 types of sensors, such as for temperature, humidity, gas, and vibration, spread across several spaces in the sensor network environment of IoT. In addition, we determined a total of 1 million tuples sent by each stream. Although the input rate is set to 1000 tuples per second, it takes 1000 seconds to receive the content of all streams, which is sufficient when conducting the experiment. Note that one stream with a total of 1 million tuples is considered to be composed of an infinite amount given that its size cannot be determined in a real environment. A total of one million tuples were defined as 300 tuples per second. An example is a sensor network environment wherein 300 sensors are spread over several spaces to collect one type of information and each sensor collects a large amount of information per second.

The key attribute values in all tuples were integers ranging from 0 to 1000. These key attribute values were generated at the same frequency across all tuples. The generation sequence was randomized; key attribute values in the sliding window were changed each time, thereby continuously varying the selection rate of the join operators.

Multiple queries were applied to the generated input streams to execute multiple join operations; each query could be applied to a maximum of 20 different input streams. Note also that because a single input stream can be present, at most once in each query, it can occur as many times as the maximum number of queries. To refine the experiment, the skewness of the input streams that appeared over several queries was reflected. In this experiment, the Zipf distribution was used to apply asymmetry to the input streams in all queries. Furthermore, in each query, one tuple from among 500, 1000, and 1500 tuples was randomly selected as the window for each input stream, and each window was generated with an even distribution. These three windows were used to create cases in which the sharing condition used

TABLE 4. Experimental variables for input streams and queries.

Parameter	Range	Description
Q	10 to 100	The total number of distinct queries
SO	0 to 1	Skewness of input streams over queries
WS	500 to 1500	Window sizes of input stream

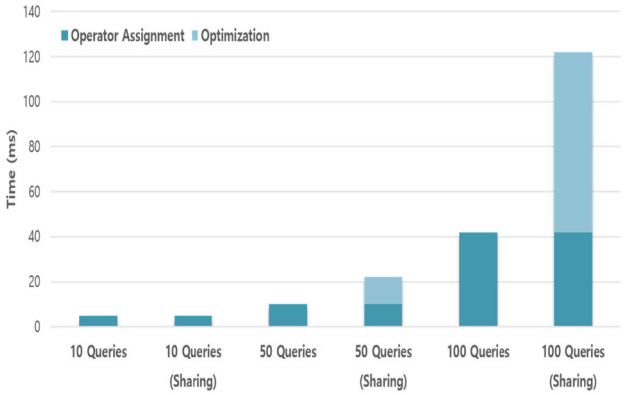


FIGURE 11. Comparison of establishment time according to number of queries (so = 0.5).

in the multiple MJoin technique was not satisfied. In this experiment, we used only the window based on the number of tuples. This is because the proposed method does not use a timestamp although the window based on time is used for window updates and routing for the tuple as a result of the combined operation.

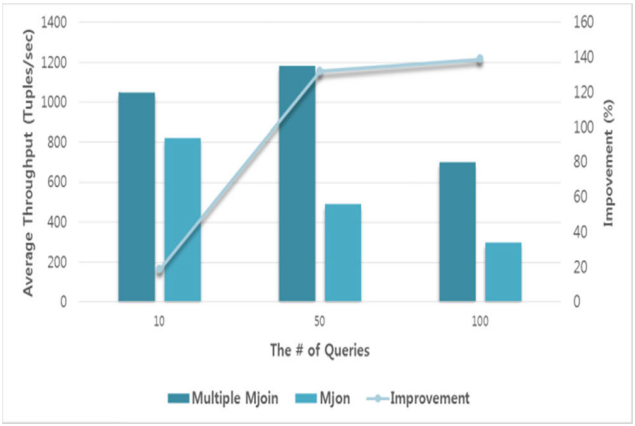
Table 4 lists the experimental parameters used in this experiment.

1) A-1. THE COMPARISON REGARDING THE TIME REQUIRED TO ESTABLISH THE GLOBAL SHARED QUERY EXECUTION PLANS

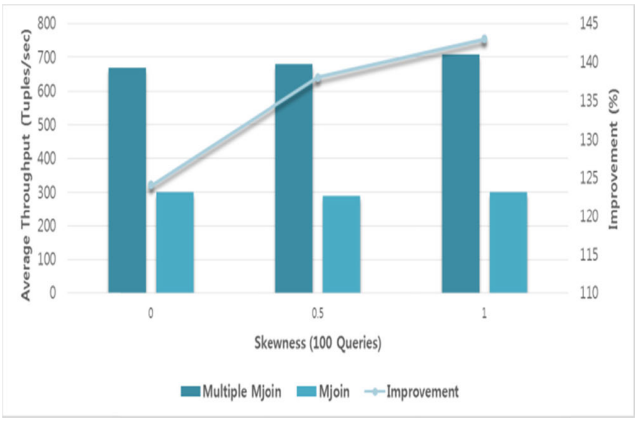
In the comparison regarding the time required to establish the global shared query execution plans, the experimental parameter SO was set to 0.5. Figure 11 shows the optimization time and operator assignment time for executing independent MJoin and multiple MJoin when the asymmetry of the input streams in each query was 0.5. The time to establish the global shared query execution plan is measured as follows.

Assuming that the allowable time to perform operator assignment for numerous queries entered by the user is t_{assign} and that the calculation time by the algorithm in Figure 5 is $t_{optimizing}$, the establishment time of the global shared query execution plan, the multiple-query setup time, is $t_{assign} + t_{optimizing}$.

Because the optimization algorithm of multiple MJoin has the effect of reducing the number of operators, it takes slightly less time than the original operator assignment used to execute independent MJoin. However, as indicated in the multiple MJoin 11 example, the operations for additional optimization increase as the number of queries increases, thereby increasing the required time.



(a) COMPARISON OF THROUGHPUT BY NUMBER OF QUERIES (SO = 0.5)



(b) COMPARISON OF THROUGHPUT BY ASYMMETRY (Q =100)

FIGURE 12. Average throughput comparison.

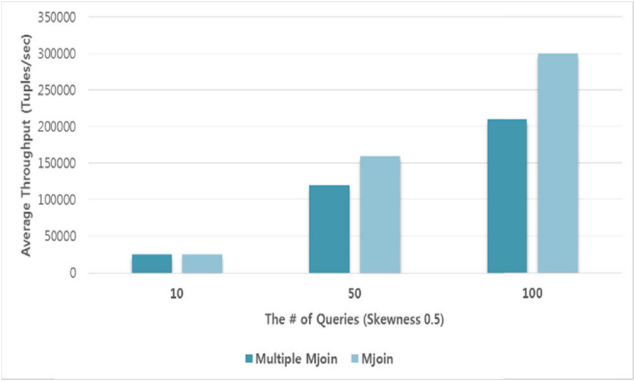
2) A-2. THROUGHPUT OF MJION AND MULTIPLE MJOIN TECHIQUES

Figure 12(a) shows the average throughput when SO is fixed at 0.5 with variations in the number of queries. The measurement unit for average throughput is defined as follows.

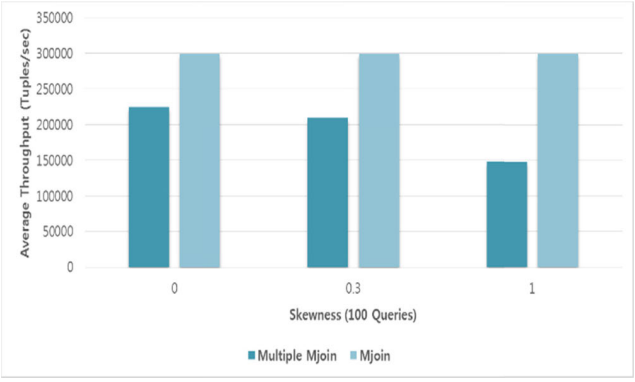
Assuming that the total number of result tuples that all queries send out from i seconds to $i + 1$ seconds is $throughput_i$, the total measured time is t_{total}, and assuming that the total number of queries is Q, the average throughput for each query AT is $\sum_{i=0}^{t_{total}} throughput_i/(t_{total} \cdot Q)$.

As the number of queries increases, the throughput of the multiple MJoin technique is observed to increase compared to independent MJoin. This is attributed to the rise in probability that more operators can be shared as the number of queries increases.

Figure 12(b) shows the average throughput when the number of queries is set to 100 with variations in the input asymmetry. As the input asymmetry increases, the common parts within the multiple queries also increase, leading to greater sharing, therefore improving the relative average



(a) COMPARISON OF TUPLE VOLUME BY NUMBER OF QUERIES (SO = 0.5)



(b) COMPARISON OF TUPLE VOLUME BY SKEWNESS (Q = 100)

FIGURE 13. Average tuple usage comparison.

throughput of the multiple MJoin technique. Consequently, if the number of queries and asymmetry of the input stream increase, then the cases in which the queries share a containment relationship increases, thereby increasing the operators shared in the multiple MJoin technique.

The two graphs in Figure 12 show no large throughput improvements after the number of queries or input asymmetry reaches a certain level. This is because the operators are already sufficiently shared given a certain number of queries and level of input asymmetry.

3) A-3. TUPLE USAGE OF MJOIN AND MULTIPLE MJOIN TECHNIQUES

Figure 13 shows the average tuple usage when SO is set to 0.5 and the number of queries is varied and when the number of queries is set to 100 and SO is varied. The measurement unit for tuple usage is as follows.

Assuming that the number of tuples in one join operator op_i is st_i and that the total number of all join operators is op_{total}, then the number of tuples maintained in the memory by all join operators registered in the system at a given point in time is $\sum_{i=1}^{op_{total}} st_i$.

As the number of queries increases and as the input asymmetry increases, the relative tuple usage by the multiple MJoin technique gradually decreases. As explained by the previous experiment, this is because more operators

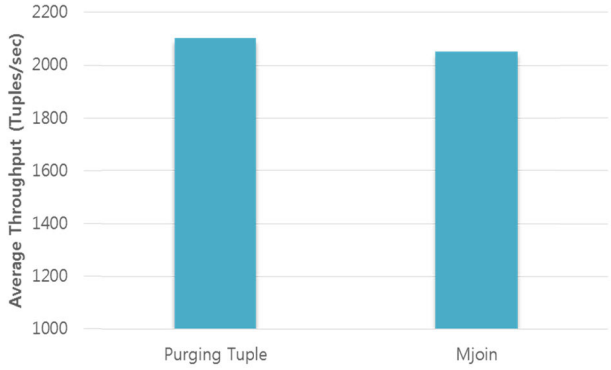


FIGURE 14. Average throughput: A ⋈ B ⋈ C VS (A ⋈ B) ⋈ C.

are shared as the number of queries and input asymmetry increase. The reduction in tuple usage when operators are shared is ensured by the sharing condition defined in this study.

4) A-4. ADDITIONAL COSTS OF THE MULTIPLE MJOIN TECHNIQUE

Because operators are not shared when independent MJoin is executed, all window updates are executed only for the raw stream, and routing is not required. Hence, no additional calculations are needed for purging tuples and dead tuples. However, in the multiple MJoin technique, by sharing operators, a join tree is formed, and purging tuples and dead tuples are used for window updating and routing of the join operation results in the tree. These two processes may offset the benefits gained by sharing operators and were, therefore, evaluated.

Figure 14 shows the results of using purging tuples when a join tree of A ⋈ B ⋈ C(MJoin) and (A ⋈ B) ⋈ C is formed in an environment using a window with 1000 tuples for all inputs. Slightly higher throughput than MJoin is exhibited when applying purging tuples because the A ⋈ B calculation to be executed in MJoin each time the join tree is formed is already cached. Thus, the additional computational cost of the purging tuples is not large enough to offset the computational gain obtained by sharing operators. Dead tuples are also used in situations in which operators are shared, and it is clear that the additional computational cost of the dead tuples alone is not large enough to offset the gain achieved by processing multiple operators at once. This is because dead tuples are generated in proportion to the existing number of shared operators.

Thus, despite the use of purging tuples and dead tuples in the multiple MJoin technique, this method exhibits vastly superior throughput and lower memory usage compared to independent MJoin, as well as relatively low additional computational costs introduced by the addition of purging tuples and dead tuples.

The multiple MJoin technique has the following characteristics.

First, the time required to establish a global shared query execution plan in the data stream is substantially reduced.

The multiple MJoin technique conducts searches based only on a query and fully accounts for the constraint placed on the sliding window, enabling the establishment of a query execution plan appropriate for a data stream environment. The experimental results demonstrated that the multiple MJoin technique achieves high improvements in performance because it conducts relatively few searches.

Second, an accurate window updating technique under the shared query execution plan was proposed. When a global shared query execution plan is established, MJoin is split and transformed into a tree or graph, leading to problems in window updating for the operation result tuples. This paper proposed an index-based search applicable to both time-based and tuple count-based windows. Moreover, this study extended the index assignment scheme and situation data structure inside the join operator. Given this framework, a purging tuple technique was proposed that maintains window update accuracy and supports conducting only a few searches.

Third, an accurate routing technique under the shared query execution plan was proposed. Routing is closely related to window updating for the join operation tuples. As such, the index information used in the window update is utilized for searching. This also allows the dead attribute required for routing, thereby supporting fast routing for the operation results. The dead tuple technique adds routing information to the join operation results and is very similar to the purging tuple technique. Because an index-based search is performed, the routing information is correctly added to the corresponding tuple. Numerous queries must be processed in an IoT environment. Systems accessed by many users require faster data stream processing. In this respect, the multiple MJoin technique is expected to enable faster processing in such environments.

5) A-5. SUMMARY

Multiple Mjoin consumes a large amount of time to set up query execution plans when compared to an independently executed Mjoin. It can be applied to real environments so that it can generate significant effects in terms of sacrificing the initial short time by considering the subsequent throughput. However, since the initial set-up time cannot process all inputs, they should be placed in the system buffer. Otherwise, all tuples inputted during this time should be removed. That is, a relatively short set-up time acts as a factor in improving the quality of the results. The multiple Mjoin technique can be evaluated as a more suitable technique in terms of a data stream environment given that it shows a shorter set-up time than the conventional optimization method.

Through various experiments, we confirmed that the multiple Mjoin technique had a better performance and used less tuples than the independently executed Mjoin. Although the probability that operators are shared is very small, considering that the number of queries is few or that the skewness of the input stream is low, it can nevertheless guarantee the same or better performance than the independently executed Mjoin. The reason is that the query execution plan is set up based on the queries of the multiple Mjoin technique.

Moreover, the performance and tuple usage of multiple Mjoin can be directly affected by the shared ratio of operators. When more queries are used and the input asymmetry becomes larger, the throughput increases and the total number of tuples used decreases because more operators are shared. However, if the number of queries and input skewness of a certain level is exceeded, we cannot expect a large increase in the processing improvement rate. The reason is that the number of tuples used and the degree of input asymmetry significantly influence operator sharing. Moreover, once it exceeds a certain level, conditions are met wherein many operators can be shared.

TABLE 5. Confusion matrix.

		Actual Value	
		Positives	Negatives
Predicted Value	Positives	TP (True Positives)	FN (False Negative)
	Negatives	FN (False Negative)	TP (True Positives)
True Positive Rate (TPR)		$\text{TPR} = \frac{TP}{P} = \frac{TP}{TP + FN}$	
False Positive Rate (FPR)		$\text{FPR} = \frac{FP}{N} = \frac{FP}{FP + TN}$	
Accuracy		$\text{Accuracy} = \frac{TN + TP}{P + N}$	
Error Rate		$\text{Error Rate} = 1 - \text{Accuracy}$	
Sensitivity = Recall		Sensitivity = True Positive Rate (TPR)	
Specificity		$\text{Specificity} = \frac{TN}{N} = \frac{TN}{FP + TN} = 1 - FPR$	

TABLE 6. Polynomial confusion matrix.

d	TP	TN	FP	FN	Sensitivity	Specificity	Accuracy	TPR	FPR
1	65	90	0	25	0.72	1.00	0.86	0.72	0.00
2	70	90	0	20	0.78	1.00	0.89	0.78	0.00
3	70	90	0	20	0.78	1.00	0.89	0.78	0.00
4	74	90	0	16	0.82	1.00	0.91	0.82	0.00
5	78	90	0	12	0.87	1.00	0.93	0.87	0.00
6	79	90	0	11	0.88	1.00	0.94	0.88	0.00
7	79	90	0	11	0.88	1.00	0.94	0.88	0.00
8	80	90	0	10	0.89	1.00	0.94	0.89	0.00
9	80	90	0	10	0.89	1.00	0.94	0.89	0.00

B. SVM OPTIMAL KERNEL FUNCTION SELECTION

This study analyzed SVM kernel functions and parameters to achieve optimal performance for the SVM classification algorithm. Table 5 lists several performance measurements that can be derived using true positive (TP), true negative (TN), false positive (FP), and false negative (FN) values in the confusion matrix of the test set results. Using Table 5, the kernel function achieving the highest accuracy was selected among the SVM kernel functions. Accuracy in this case expresses how closely the actual value after SVM training matches the predicted value of training.

Tables 6, 7, and 8 show the confusion matrices of the polynomial kernel, RBF kernel, and sigmoid kernel functions,

TABLE 7. RBF confusion matrix.

γ	TP	TN	FP	FN	Sensitivity	Specificity	Accuracy	TPR	FPR
0.1	83	90	0	7	0.92	1.00	0.96	0.92	0.00
0.2	80	90	0	10	0.89	1.00	0.94	0.89	0.00
0.3	75	90	0	15	0.83	1.00	0.92	0.83	0.00
0.4	70	90	0	20	0.78	1.00	0.89	0.78	0.00
0.5	69	90	0	21	0.77	1.00	0.88	0.77	0.00
0.6	63	90	0	27	0.70	1.00	0.85	0.70	0.00
0.7	60	90	0	30	0.67	1.00	0.83	0.67	0.00
0.8	58	90	0	32	0.64	1.00	0.82	0.64	0.00
0.9	50	90	0	40	0.56	1.00	0.78	0.56	0.00
1.0	50	90	0	40	0.56	1.00	0.78	0.56	0.00

TABLE 8. Sigmoid confusion matrix.

γ, C	TP	TN	FP	FN	Sensitivity	Specificity	Accuracy	TPR	FPR
0.1, 0.1	70	90	0	20	0.78	1.00	0.89	0.78	0.00
...	...	...	...	...	...	...	...	...	...
0.1, 0.5	86	90	0	4	0.96	1.00	0.98	0.96	0.00
0.1, 0.6	89	90	0	1	0.99	1.00	0.99	0.99	0.00
0.1, 0.7	87	90	0	3	0.97	1.00	0.98	0.97	0.00
...	...	...	...	...	...	...	...	...	...
0.2, 0.1	90	90	0	0	1.00	1.00	1.00	1.00	0.00
0.2, 0.2	90	80	10	0	1.00	0.89	0.94	1.00	0.11
0.2, 0.3	90	73	17	0	1.00	0.81	0.91	1.00	0.19
...	...	...	...	...	...	...	...	...	...
0.3, 0.1	60	80	10	30	0.67	0.89	0.78	0.67	0.11
0.3, 0.2	68	80	10	22	0.76	0.89	0.82	0.76	0.11
0.3, 0.3	60	60	30	30	0.67	0.67	0.67	0.67	0.33
...	...	...	...	...	...	...	...	...	...
0.4, 0.2	60	62	28	30	0.67	0.69	0.68	0.67	0.31
0.4, 0.3	69	70	20	21	0.77	0.78	0.77	0.77	0.22
0.4, 0.4	67	64	26	23	0.74	0.71	0.73	0.74	0.29
...	...	...	...	...	...	...	...	...	...
0.5, 0.3	59	60	30	31	0.66	0.67	0.66	0.66	0.33
0.5, 0.4	68	65	25	22	0.76	0.72	0.74	0.76	0.28
0.5, 0.5	60	60	30	30	0.67	0.67	0.67	0.67	0.33
...	...	...	...	...	...	...	...	...	...
0.6, 0.7	60	60	30	30	0.67	0.67	0.67	0.67	0.33
0.6, 0.8	60	60	30	30	0.67	0.67	0.67	0.67	0.33
0.6, 0.9	58	60	30	32	0.64	0.67	0.66	0.64	0.33
...	...	...	...	...	...	...	...	...	...
0.7, 0.7	60	70	20	30	0.67	0.78	0.72	0.67	0.22
0.7, 0.8	60	64	26	30	0.67	0.71	0.69	0.67	0.29
0.7, 0.9	60	60	30	30	0.67	0.67	0.67	0.67	0.33
...	...	...	...	...	...	...	...	...	...
0.8, 0.1	58	60	30	32	0.64	0.67	0.66	0.64	0.33
0.8, 0.2	58	60	30	32	0.64	0.67	0.66	0.64	0.33
0.8, 0.3	52	60	30	38	0.58	0.67	0.62	0.58	0.33
...	...	...	...	...	...	...	...	...	...
0.9, 0.1	58	60	30	32	0.64	0.67	0.66	0.64	0.33
0.9, 0.2	57	60	30	33	0.63	0.67	0.65	0.63	0.33
0.9, 0.3	51	60	30	39	0.57	0.67	0.62	0.57	0.33

respectively. The polynomial kernel function achieved an accuracy of approximately 94% when d was 6 or higher. The RBF kernel function achieved the highest accuracy of 96% when γ was 0.1, and as γ increased, the accuracy gradually decreased. In the sigmoid kernel function analysis, the ranges of γ and C were each set to 0.1–0.9, and a GS was performed at intervals of 0.1 within this range. The accuracy increased as γ approached 0.1; however, the best accuracy was achieved when γ was equal to 0.2. Furthermore, the highest accuracy of 100% was achieved by the sigmoid function when γ was 0.2 and C was 0.1. Hence, the sigmoid kernel function was selected as the optimal kernel function for the SVM classification algorithm.

TABLE 9. Error rate according to window size change.

Window Size	REMS (%)	MAE (%)	MAPE (%)
1000	2.93	2.70	12.49
2000	2.67	2.44	12.12
3000	2.35	2.12	11.87
4000	2.13	1.91	10.79
5000	2.01	1.77	10.56
Average	2.42	2.19	11.57

C. PERFORMANCE EVALUATION ACCORDING TO THE CHANGES IN THE SLIDING WINDOW SIZE

The experiment measured the accuracies of queries in the TinyDB. A total of 10 nodes were used, including one sink and nine intermediate nodes. Every 5 s, the data calculated for temperature, humidity, gas, vibration, and detection were sent to the stream data storage.

MJoin queries were performed on the collected data during stream data management, and the data were saved in the TinyDB via the SVM algorithm's data classification and reduction process.

It was necessary to measure the error rate, because the experiment used irregular data, which reflected the real world, rather than a linear relationship. The experiments conducted in this study measured the error rate according to changes in the sliding window size. The root mean square error (RMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE) were used to measure the error rates of the experiment. These errors are defined in Equations 5, 6, and 7, respectively.

$$\mathrm{REMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Z_t - F_t)^2} \tag{5}$$

$$\mathrm{MAE} = \frac{1}{n}\sum_{t=1}^{n}|Z_t - F_t| \tag{6}$$

$$\mathrm{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{Z_t - F_t}{Z_t}\right| \times 100\% \tag{7}$$

In these equations, n is the number of samples used to fit the data, Z_t is the actual value at time t, and F_t is the predicted value for each model.

Table 9 presents the results of measuring the error rate according to changes in the window size. The average RMSE was 2.42%, MAE was 2.19%, and MAPE was 11.57%. According to the evaluation criteria for MAPE in Table 10,

TABLE 10. Evaluation of Mean Absolute Percentage Error (MAPE) value.

MAPE	Evaluation
0% ≤ MAPE < 10%	Very accurate predictions.
10% ≤ MAPE < 20%	Relatively accurate predictions
20% ≤ MAPE < 30%	Relatively rational prediction
50% ≤ MAPE	Inaccurate prediction

TABLE 11. Reduction result according to window size change.

Window Size	Reduction Rate (%)
1000	17.1
2000	17.4
3000	17.2
4000	17.5
5000	18.7
Average	17.58

TABLE 12. Classification accuracy result according to window size change.

Window Size	Accuracy (%)
1000	82.2
2000	84.4
3000	88.6
4000	86.7
5000	87.8
Average	85.94

11.57% indicates a relatively accurate prediction, demonstrating the accuracy of the system proposed in this study.

Tables 11 and 12 show the reduction ratio and classification accuracy, respectively, according to changes in the window size. In the experimental data of this study, each stream's ratio was the same, and only the window size was varied as the measurements were taken. In total, 15,593 data groups were used, and the window size was divided into 1000, 2000, 3000, 4000, and 5000 according to the number of tuples. The SVM algorithm was used to reduce the data.

In the experiment results, the storage space was reduced by a maximum of 18.7% when the window size was divided by 5000, which was found to be the most efficient. As the window size became larger, this reduction increased. Furthermore, the classification accuracy was highest at 88.6%, when the window size was divided by 3000.

This paper also compared the performance of the SVM classification algorithm with other algorithms. In the SVM classification algorithm, optimal kernel functions were used for the other algorithms and subsequent experiments were conducted. As shown in Figure 15, Decision Tree, Naive Bayes, SVM, k-nearest neighbor, and multi-layer perceptron algorithms yielded accuracies of 72.83%, 70.98%, 85.94%, 74.43%, and 76.85%, respectively. The experimental results demonstrate that the SVM algorithm using the optimal kernel achieves at least 9% higher classification accuracy than the other algorithms.

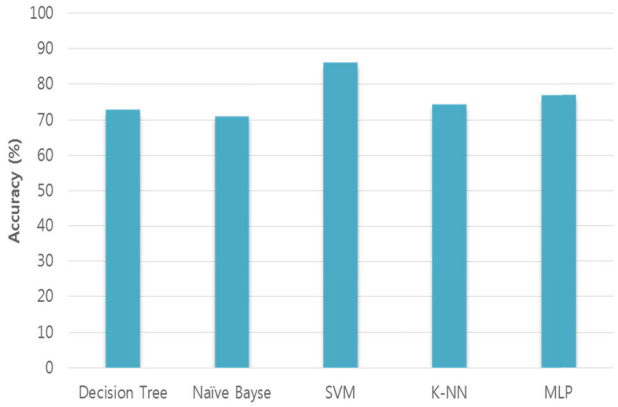


FIGURE 15. Classification performance comparison by classification algorithm.

D. IMPLEMENTATION OF IoT-BASED SMART HOME SYSTEM

Finally, the classified and reduced sensor data were used to implement the IoT-based smart home system. Figure 16 shows the results of implementing the IoT-based smart home system. This was designed such that a warning message can be sent when a dangerous situation occurs, so that the user can respond.

The system application is an important module, which allows the administrator to monitor and remotely control the device statuses and sensor data that are measured by the Arduino (Uno) module. This was designed to allow control even in environments using old versions of Android, and to classify the minimum amount of data that the administrator must receive within the home so that it is easy for the user to use.

The user can use the application to observe the current operating statuses of devices and to change these. The current operational statuses of device stored in the database are sent to the application, and changes in device operations can be observed.

The user can use these monitoring features to find unnecessary tasks being performed by the devices within the home. In addition, the user can use the application to set three device statuses: On, Off, and Wait. When On is selected, the device operates even when a linked task is being performed, because the user's control commands are given priority over the task commands. When Off is selected, the user's commands are again given priority, and the device's operations are paused.

The Wait status changes the device's status such that manual control of the device by the user is paused, and the device is included in the task and operates automatically. In this

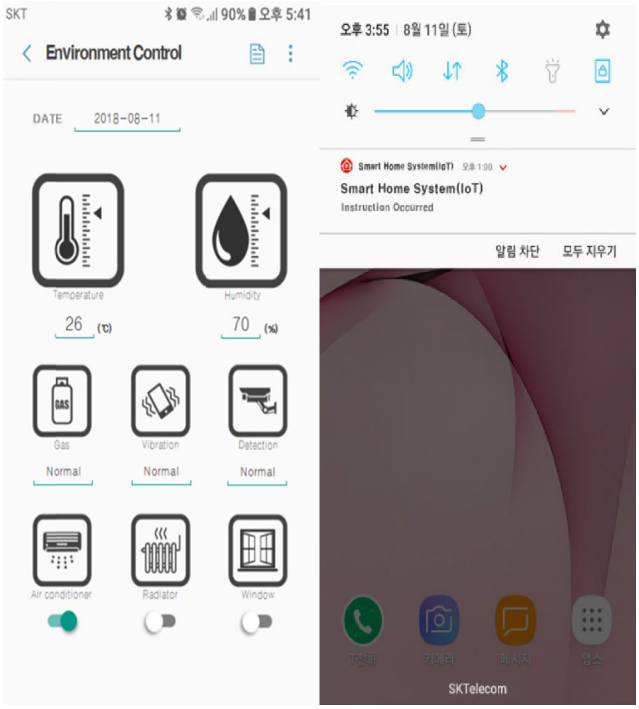


FIGURE 16. The implementation result of smart home system based on internet things.

manner, the user can receive customized services from the automatically operating system.

When the gas sensor value exceeds the threshold value, the system determines that a gas leak has occurred in the home and transmits a warning message created by FCM to the user. After a gas leak is detected, the temperature sensor values are extracted and compared with the temperature threshold value to determine if there is a fire.

If a fire is detected, then a fire warning message is sent to the user so that they can deal with the fire. By providing the user with this two-stage dangerous situation notification system, the user can quickly respond to dangerous situations that occur within the home.

The data are used to perform the task and device classification of each sensor into quarters and seasons, and the tasks are selected.

The sensor's threshold values are used to determine the task operations. This is to prevent interference and collisions between tasks, because the relationships between the selected devices are set automatically, and they operate in sequence. In addition, this is to classify the tasks of the temperature and humidity sensors by season, and to select the operating devices that are included in the tasks.

Table 13 shows the tasks corresponding to each sensor.

The temperature sensor's tasks consist of temperature tasks and window tasks for each season. The window tasks set a relationship according to the temperature sensor's tasks, and they open and close windows.

The humidity sensor's tasks consist of humidity tasks for each season. The gas sensor's task is the window task.

TABLE 13. Each sensor task.

Type	Composition
Temperature Sensor	Temperature Task (Spring)
	Temperature Task (Summer)
	Temperature Task (Fall)
	Temperature Task (Winter)
	Window Task
Humidity Sensor	Humidity Task (Spring)
	Humidity Task (Summer)
	Humidity Task (Fall)
	Humidity Task (Winter)
Gas Sensor	Window Task

TABLE 14. Temperature sensor task devices.

Type	Composition
Temperature (Spring)	Fan, Air Conditioner, Radiator, Heater
Temperature (Summer)	Fan, Air Conditioner
Temperature (Fall)	Fan, Air Conditioner, Radiator, Heater
Temperature (Winter)	Radiator, Heater

This involves opening and closing the windows according to whether or not gas is present.

In the task priorities, the window tasks have the highest priority. When these tasks begin, a message stating that window operations are complete is sent, and then the task with the next highest priority is performed.

In addition, when a gas leak is detected all currently operating devices are paused, and the window opening task is performed. In this manner, tasks are performed in sequence by classifying tasks and setting their relationships and priorities. In addition, interference between tasks, collisions between devices, and bottlenecks can be minimized.

Table 14 shows the classification of devices corresponding to the temperature sensor task.

By classifying devices corresponding to tasks for each season and completely excluding the operation of devices that are not used each season, device malfunctions and electricity waste can be minimized. Spring and fall tasks have large daily temperature ranges and large differences between the first and last temperature. Therefore, all devices utilized for temperature adjustment tasks are used.

Table 15 shows the device classification of the humidity sensor.

Like the temperature sensors, the humidity sensors also set the devices differently that are included in tasks according to the season. The temperature and humidity sensors compare their values with the threshold values for each season

TABLE 15. Humidity sensor task devices.

Type	Composition
Temperature (Spring)	Humidifier, Dehumidifier
Temperature (Summer)	Dehumidifier
Temperature (Fall)	Humidifier, Dehumidifier
Temperature (Winter)	Dehumidifier

and begin the tasks and select and operate the devices that correspond to these tasks.

By setting the task priorities, the tasks that are being performed and their priorities can be compared, and higher-priority tasks can suspend existing tasks, while lower-priority tasks can be performed after existing tasks.

In this manner, the user can freely intervene while tasks are being performed to change the statuses of devices and perform manual control. By doing so, user-customized services can be provided to the user, and the user can experience greater convenience via tasks that operate accurately according to the season.

V. CONCLUSION

Current systems in IoT environments use a large amount of various sensor data. The types of sensors vary, because smart devices are used differently according to the location, and methods for using the sensors also vary according to their purposes. However, most existing studies have focused on cooperation between devices or task efficiency, and there is a tendency for these systems to have manual operations, because they have not created an environment with sensors or other media.

Smart home systems must be able to create relationships between various sensors attached to devices within the home and link them to efficiently perform tasks such as energy management, cooling, heating, and ventilation. Various sensor data must be collected, and monitoring services must be provided to the user.

To process fast and continuous stream data collected from sensor networks in an IoT environment, the hash table-window join operator multiple MJoin was used to optimize queries, and the SVM algorithm was used to classify and reduce data for the purpose of efficiently managing stream data storage. A global shared query execution technique for the query optimization of multiple MJoin was used, and experiments verified that it yielded notable improvements in performance with relatively few searches. This study evaluated the system performance through experimentation according to changes in sliding window size and the selected optimal kernel function of the SVM classification algorithm through evaluations of different kernel functions for efficient storage management of the stream data. Based on the performance evaluation results, the sigmoid kernel function was selected as the optimal kernel function for the SVM classification algorithm. According to the SVM classification algorithm results based on changes in sliding window size, the average error rate was 2.42%, the reduction result was 17.58%, and the classification accuracy was 85.94%. Based on the comparison of the SVM classification performance with that of other algorithms, the SVM classification algorithm achieved a minimum 9% better classification performance than the other classification algorithms evaluated.

In addition, this study has proposed an IoT-based smart home system that can use the classified and reduced sensor data to intelligently control devices within the home.

The results of experiments on the classification and reduction techniques proposed in this study demonstrated that when the window size was divided by 5000, the storage space was reduced by a maximum of 18.7%, which was the most efficient. It was found that as the window size became larger, this reduction increased. The classification accuracy was highest at 88.6% with the window size divided by 3000.

In this study, we conducted experiments by arbitrarily assuming 20 IoT environment sensors to optimize the multiple Mjoin query processing. Given that various sensor data are generated without noticeable delay for a home system in the IoT environment, a method that can efficiently process a large amount of sensor data is necessary. Therefore, based on comparison of the conventional Mjoin and multiple Mjoin methods, we proposed a method of optimizing the query processing of stream data. In addition, to address the situation of the smart home system based on the IoT, we classified the sensor data (temperature, humidity, and gas) measured through the Arduino module and date data into tasks using a decision tree. Then, we set up the devices using the sensor data. Furthermore, we designed the system to intelligently control the priorities for ventilation, temperature, fire, and break-in using five sensors. Considering that the designed system was implemented using only the limited sensor data measured through the Arduino module, several areas did not reflect all the environments of the home system in the IoT environment. Thus, in the future, if we use the classified and reduced sensor data proposed in this study using various environmental sensors, the efficiency and convenience of the system are expected to increase because we will be able to configure an environment with a more intelligent system than the conventional home system, thus meeting users' requirements.

In future research, it will be necessary to study a more efficient sensor data-processing algorithm, which takes the processing time into account. Future studies will also expand on and refine the classification statuses of the decision-making tree proposed in this study. In addition, future studies will use this system in a variety of IoT environments other than a home system, to create more convenient and efficient IoT environments regardless of location.

REFERENCES

[1] M.-Z. Song, "A study on business types of IoT-based smarthome: Based on the theory of platform typology," *J. Inst. Internet, Broadcast. Commun.*, vol. 16, no. 2, pp. 27–40, 2016, doi: 10.7236/JIIBC.2016.16.2.27.

[2] S. Yoon and J. Kim, "A study on the user's value of the smart home service in the Internet of Things technology," *Int. J. Future Gener. Commun. Netw.*, vol. 10, no. 6, pp. 65–80, Jun. 2017, doi: 10.14257/ijfgcn.2017.10.6.07.
[3] M. Alaa, A. A. Zaidan, B. B. Zaidan, M. Talal, and M. L. M. Kiah, "A review of smart home applications based on Internet of Things," *J. Netw. Comput. Appl.*, vol. 97, pp. 48–65, Nov. 2017, doi: 10.1016/j.jnca.2017.08.017.
[4] M. J. Lee, J. S. Lee, and Y. S. Han, "Adaptive priority queue-driven task scheduling for sensor data processing in IoT environments," *J. Korea Multimedia Soc.*, vol. 29, no. 9, pp. 1559–1566, 2017, doi: 10.9717/kmms.2017.20.9.1559.
[5] Y. Yin, J. Xia, Y. Li, Y. Xu, W. Xu, and L. Yu, "Group-wise itinerary planning in temporary mobile social network," *IEEE Access*, vol. 7, pp. 83682–83693, 2019, doi: 10.1109/ACCESS.2019.2923459.
[6] W. Lee, S. Cho, P. Chu, H. Vu, S. Helal, W. Song, Y.-S. Jeong, and K. Cho, "Automatic agent generation for IoT-based smart house simulator," *Neurocomputing*, vol. 209, pp. 14–24, Oct. 2016, doi: 10.1016/j.neucom.2015.04.130.
[7] Y. Yin, L. Chen, Y. Xu, J. Wan, H. Zhang, and Z. Mai, "QoS prediction for service recommendation with deep feature learning in edge computing environment," *Mobile Netw. Appl.*, vol. 25, no. 2, pp. 391–401, Apr. 2020, doi: 10.1007/s11036-019-01241-7.
[8] P. P. Ray, M. Mukherjee, and L. Shu, "Internet of Things for disaster management: State-of-the-Art and prospects," *IEEE Access*, vol. 5, pp. 18818–18835, 2017, doi: 10.1109/ACCESS.2017.2752174.
[9] W.-Y. Lee, H.-M. Ko, J.-H. Yu, and K.-B. Sim, "An implementation of smart dormitory system based on Internet of Things," *J. Korean Inst. Intell. Syst.*, vol. 26, no. 4, pp. 295–300, Aug. 2016, doi: 10.5391/JKIIS.2016.26.4.295.
[10] D. Mulfari, A. L. Minnolo, and A. Puliafito, "Wearable devices and IoT as enablers of assistive technologies," in *Proc. 10th Int. Conf. Develop. eSystems Eng. (DeSE)*, Jun. 2017, pp. 14–19, doi: 10.1109/DeSE.2017.51.
[11] J.-H. Lee, "Energy-efficient clustering scheme in wireless sensor network," *Int. J. Grid Distrib. Comput.*, vol. 11, no. 10, pp. 103–112, Oct. 2018, doi: 10.14257/ijgdc.2018.11.10.09.
[12] H. Gao, Y. Duan, L. Shao, and X. Sun, "Transformation-based processing of typed resources for multimedia sources in the IoT environment," *Wireless Netw.*, pp. 1–17, Nov. 2019, doi: 10.1007/s11276-019-02200-6.
[13] T.-Y. Kim, S.-H. Bae, and Y.-E. An, "A study on intelligent smart home system in Internet of Things (IoT) environment," *Int. J. Internet Things Appl.*, vol. 3, no. 1, pp. 1–6, 2019.
[14] D.-W. Song, K.-S. Kim, and S.-K. Lee, "An relational analysis between humidity, temperature and fire occurrence using public data," *Fire Sci. Eng.*, vol. 28, no. 2, pp. 82–90, Apr. 2014, doi: 10.7731/KIFSE.2014.28.2.082.
[15] J. Geng and Z. He, "Innovation and development strategy of logistics service based on Internet of Things and RFID automatic technology," *Int. J. Future Gener. Commun. Netw.*, vol. 9, no. 12, pp. 251–262, Dec. 2016, doi: 10.14257/ijfgcn.2016.9.12.23.
[16] H. Yang, "The novel modern Internet of Things system structure optimization methodology based on information theory and communication signal transmission model," *Int. J. Future Gener. Commun. Netw.*, vol. 9, no. 9, pp. 119–132, Sep. 2016, doi: 10.14257/ijfgcn.2016.9.9.11.
[17] P. Kumar and U. C. Pati, "IoT based monitoring and control of appliances for smart home," in *Proc. IEEE Int. Conf. Recent Trends Electron., Inf. Commun. Technol. (RTEICT)*, Bangalore, India, May 2016, pp. 1045–1050, doi: 10.1109/RTEICT.2016.7808011.
[18] S. Kwon, D. Park, H. Bang, and Y. Park, "Real-time and parallel semantic translation technique for large-scale streaming sensor data in an IoT environment," *J. KIISE*, vol. 42, no. 1, pp. 54–67, Jan. 2015, doi: 10.5626/JOK.2015.42.1.54.
[19] C. Imtar, U. Muhammad, F. Arshad, and U. K. Wajid, "Towards the development of an efficient and cost effective intelligent home system based on the Internet of Things," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 6, pp. 343–350, 2016.
[20] K. Nair, J. Kulkarni, M. Warde, Z. Dave, V. Rawalgaonkar, G. Gore, and J. Joshi, "Optimizing power consumption in iot based wireless sensor networks using Bluetooth low energy," in *Proc. Int. Conf. Green Comput. Internet Things (ICGCIoT)*, Oct. 2015, pp. 589–593, doi: 10.1109/ICGCIoT.2015.7380533.
[21] C. K. Dehury and P. K. Sahoo, "Design and implementation of a novel service management framework for IoT devices in cloud," *J. Syst. Softw.*, vol. 119, pp. 149–161, Sep. 2016, doi: 10.1016/j.jss.2016.06.059.
[22] M. Tao, K. Ota, and M. Dong, "Ontology-based data semantic management and application in IoT- and cloud-enabled smart homes," *Future Gener. Comput. Syst.*, vol. 76, pp. 528–539, Nov. 2017, doi: 10.1016/j.future.2016.11.012.
[23] W. Lee, S. Cho, P. Chu, H. Vu, S. Helal, W. Song, Y.-S. Jeong, and K. Cho, "Automatic agent generation for IoT-based smart house simulator," *Neurocomputing*, vol. 209, pp. 14–24, Oct. 2016, doi: 10.1016/j.neucom.2015.04.130.
[24] T. Tachibana, T. Furuichi, and H. Mineno, "Implementing and evaluating priority control mechanism for heterogeneous remote monitoring IoT system," in *Proc. Adjunct Proc. 13th Int. Conf. Mobile Ubiquitous Systems: Comput. Netw. Services-MOBIQUITOUS*, 2016, pp. 239–244, doi: 10.1145/3004010.3004040.
[25] H. Gao, Y. Xu, Y. Yin, W. Zhang, R. Li, and X. Wang, "Context-aware QoS prediction with neural collaborative filtering for Internet-of-Things services," *IEEE Internet Things J.*, early access, Dec. 2, 2019, doi: 10.1109/JIOT.2019.2956827.
[26] R. M. Duarte, A. R. Du Bois, M. L. Pilla, G. G. H. Cavalheiro, and R. H. S. Reiser, "Comparing the performance of concurrent hash tables implemented in haskell," *Sci. Comput. Program.*, vol. 173, pp. 56–70, Mar. 2019, doi: 10.1016/j.scico.2018.06.004.
[27] E. Brun, A. Guittet, and F. Gibou, "A local level-set method using a hash table data structure," *J. Comput. Phys.*, vol. 231, no. 6, pp. 2528–2536, Mar. 2012, doi: 10.1016/j.jcp.2011.12.001.
[28] Y. Zhu, V. Raghavan, and E. A. Rundensteiner, "A new look at generating multi-join continuous query plans: A qualified plan generation problem," *Data Knowl. Eng.*, vol. 69, no. 5, pp. 424–443, May 2010, doi: 10.1016/j.datak.2009.11.001.
[29] J. Lee, E. Lee, and D.-K. Baik, "Simulation and performance evaluation of the self-adaptive light control system," *J. Korea Soc. Simul.*, vol. 25, no. 2, pp. 63–74, Jun. 2016, doi: 10.9709/JKSS.2016.25.2.063.
[30] J. Yu, J. Li, Z. Yu, and Q. Huang, "Multimodal transformer with multi-view visual representation for image captioning," *IEEE Trans. Circuits Syst. Video Technol.*, early access, Oct. 15, 2019, doi: 10.1109/TCSVT.2019.2947482.
[31] J. Yu, B. Zhang, Z. Kuang, D. Lin, and J. Fan, "IPrivacy: Image privacy protection by identifying sensitive objects via deep multi-task learning," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1005–1016, May 2017, doi: 10.1109/TIFS.2016.2636090.
[32] M. Zięba, J. M. Tomczak, M. Lubicz, and J. Świątek, "Boosted SVM for extracting rules from imbalanced data in application to prediction of the post-operative life expectancy in the lung cancer patients," *Appl. Soft Comput.*, vol. 14, pp. 99–108, Jan. 2014, doi: 10.1016/j.asoc.2013.07.016.
[33] Z. Yang and Y. Xu, "A safe sample screening rule for Laplacian twin parametric-margin support vector machine," *Pattern Recognit.*, vol. 84, pp. 1–12, Dec. 2018.
[34] J. Yu, Z. Kuang, B. Zhang, W. Zhang, D. Lin, and J. Fan, "Leveraging content sensitiveness and user trustworthiness to recommend fine-grained privacy settings for social image sharing," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1317–1332, May 2018.
[35] S. Ahmad, S. Malik, and D.-H. Kim, "Comparative analysis of simulation tools with visualization based on realtime task scheduling algorithms for IoT embedded applications," *Int. J. Grid Distrib. Comput.*, vol. 11, no. 2, pp. 1–10, Feb. 2018, doi: 10.14257/ijgdc.2018.11.2.01.
[36] Y. Yin, L. Chen, Y. Xu, and J. Wan, "Location-aware service recommendation with enhanced probabilistic matrix factorization," *IEEE Access*, vol. 6, pp. 62815–62825, 2018, doi: 10.1109/ACCESS.2018.2877137.
[37] J. Yu, M. Tan, H. Zhang, D. Tao, and Y. Rui, "Hierarchical deep click feature prediction for fine-grained image recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Jul. 30, 2019, doi: 10.1109/TPAMI.2019.2932058.

TAE-YEUN KIM received the B.S., M.S., and Ph.D. degrees from the Department of Computer Science and Statistics, Chosun University, Gwangju, South Korea, in 2002, 2005, and 2015, respectively. From 2012 to 2015, he was a Senior Researcher with Shinhan Systems Corporation. He is currently working as a Professor with Chosun University, South Korea. His research interests include artificial intelligence, bioinformatics, smart grid computing, and the IoT.

SANG-HYUN BAE received the B.S. and M.S. degrees from the Department of Electrical Engineering, Chosun University, Gwangju, South Korea, in 1982 and 1984, respectively, and the Ph.D. degree from the Department of Information Science, Tokyo Metropolitan University, Tokyo, Japan, in 1988. He was a Researcher with the Department of Electrical Engineering, Tokyo Institute of Technology, Japan, in 1985. He was also a Visiting Professor with the Department of Information Engineering, Nara Institute of Technology, Japan, in 1997, and at the Department of Information Engineering, University of Alberta, Canada, in 2002. He is currently a Professor with the Department of Computer Science and Statistics, Chosun University, Gwangju. He was a member of the Board of Directors of NRF, South Korea, from 2012 to 2013.

YOUNG-EUN AN received the Ph.D. degree in information and communication engineering from Chosun University, in 2010. She was a Professor at the Chosun University College of Science & Technology, South Korea, from 2011 to 2014. Since March 2014, she has been a Professor at Chosun University, South Korea.