The Computer Journal Advance Access published October 21, 2013

doi:10.1093/comjnl/bxt122

Efficient Iceberg Query Processing in Sensor Networks

HEEJUNG YANG AND CHIN-WAN CHUNG*

Department of Computer Science, KAIST, Daejeon 305-701, Republic of Korea
**Corresponding author: chungcw@kaist.edu*

The iceberg query finds data whose aggregate values exceed a pre-specified threshold. To process an iceberg query in sensor networks, all sensor data have to be aggregated and then sensor data whose aggregate values are smaller than the threshold are eliminated. Whether a certain sensor datum is in the query result depends on the other sensor data values. Since sensor nodes are distributed, communications between sensor nodes are required to know the sensor data from the other sensor nodes. However, sensor nodes have limited energy resources and communication is a primary source of the energy consumption. Thus, reducing the communication overhead is the most critical issue in sensor networks. In this paper, we propose an energy-efficient iceberg query processing technique in sensor networks. To compactly represent the data transmitted, a lossless sensor data compression method based on the Fundamental Theorem of Arithmetic is devised. To reduce the energy consumption caused by the number of data transmitted, a filtering-based query processing method is devised. Using the temporal correlation of sensor data and the semantics of an iceberg query, a prediction model is proposed. Based on the predicted future query result, sensor nodes effectively filter out unnecessary transmissions. The experimental results confirm the effectiveness of our approach.

Keywords: sensor networks; iceberg query; temporal correlation

Received 19 March 2013; revised 14 August 2013
Handling editor: Yannis Manolopoulos

1. INTRODUCTION

Recent advances in ubiquitous computing have led to the emergence of wireless sensor networks. Sensor networks consist of a large number of sensor nodes and each sensor node has capabilities of sensing, processing and communication. They enable data collection from the physical environment in unprecedented scales. There are various applications for sensor networks such as environmental monitoring, industrial maintenance and battlefield surveillance. Sensor networks continuously generate a large amount of sensor data and not all data may be of interest to users. For extracting meaningful information from all sensor data, iceberg queries can be used. An iceberg query performs an aggregate function over a set of attributes to find data whose aggregate values exceed a certain threshold. A key characteristic of an iceberg query is that the number of above-threshold data is very small relative to a large amount of input data [1]. The following query is an structured query language (SQL) form of an iceberg query based on the relation $R(A_1, A_2, \ldots, A_k)$ and a threshold T.

```
SELECT    A1, A2, ..., Ak, aggFunction()
FROM      R
GROUP BY  A1, A2, ..., Ak
HAVING    aggFunction() ≥ T
```

aggFunction() represents an aggregate function. Any aggregate function can be used such as COUNT, MIN, MAX, SUM and AVG or a certain user-defined aggregate function.

There is a challenge when we process iceberg queries in sensor networks. It is caused by distributed data sources. A sensor network consists of a large number of distributed sensor nodes. Sensor data are generated by these distributed sensor nodes. The query result is finely distributed across all sensor nodes so that it does not appear large at any one sensor node. Whether a certain sensor datum is in the query result depends on the other sensor data values. A naive implementation of processing iceberg queries is to use a centralized approach in which all sensor data are collected by the base station, which then applies an aggregate function over the received sensor data. However, this causes prohibitively high communication

cost. Sensor nodes are usually battery-powered and it is not easy to replace their batteries. Therefore, reducing the energy consumption is one of the most critical issues in sensor networks. Among the various tasks performed by a sensor node, radio communication is a primary source of energy consumption. Therefore, it is impractical to transmit all sensor data to the base station for query processing.

The amount of energy spent on communication depends on the size and the number of data transmitted. The size of data transmitted can be reduced by data compression techniques. However, traditional data compression techniques are not applicable to sensor networks because of the limited resources of sensor nodes and the distributed nature of data generation. Usually, data compression techniques exploit the statistical redundancy to represent data more concisely. To find out such statistical redundancy of sensor data generated from all sensor nodes, a large amount of historical data is required. Since sensor nodes are distributed, transmissions of sensor data to the base station are inevitable to gather a large amount of historical data. This causes a lot of communication overheads. Because of resource constraints of sensor networks, lossy compression techniques such as wavelets and regression have been studied to provide much higher compression rates. However, these techniques degrade the accuracy of sensor data [2]. For applications requiring fine-grained sensor data, the loss of any data is inappropriate. For such applications, it is necessary to devise a lossless data compression method while considering the resource constraints of sensor nodes.

The other way to reduce the communication overhead in sensor networks is to perform an *in-network aggregation* [3, 4]. In this approach, a routing tree rooted at the base station is first established, and the sensor data are then aggregated and collected along the routing tree to the base station. By performing computation within the network, it can reduce the number of sensor data transmitted compared with that of the centralized approach. The in-network aggregation can be used to process iceberg queries. For MIN and MAX aggregate functions, the in-network aggregation can reduce the communication overhead by selecting a local MIN or MAX value of sensor data and transmitting it to the parent sensor node. However, for aggregate functions such as COUNT, SUM and AVG, we cannot be sure about the final query result before all sensor data are aggregated. Since the in-network aggregation is performed along the routing topology, a sensor node only knows the partially aggregated results from its descendant nodes on the routing topology and does not know the partially aggregated results from its sibling nodes and its ancestor nodes on the routing topology. There may exist sensor data from the other sensor nodes that will contribute to increasing the corresponding aggregate value. Because the iceberg query finds sensor data whose aggregate values are above some specified threshold, local filtering by the in-network aggregation cannot be applied to these aggregate functions. Therefore, like the centralized approach, sensor data whose final aggregate values are smaller than the threshold should be transmitted to the base station anyway. Since these data are not to be included in the query result, transmissions of them waste the energy of sensor nodes. Thus, for iceberg queries whose aggregate functions are COUNT, SUM or AVG, we need an energy-efficient data collection method while considering the semantics of the iceberg query.

In this paper, we propose an energy-efficient iceberg query processing technique in sensor networks. COUNT, SUM and AVG are considered as aggregate functions of the iceberg queries. To reduce the size of data transmitted, a lossless data compression method is devised using prime numbers. The size of data transmitted is reduced to the size of one sensing attribute regardless of the number of sensing attributes specified in the `SELECT` clause. To reduce the number of transmissions, a filtering-based data collection method is devised. Since sensor data generally have strong temporal correlations, a large portion of the previous query result will be generated later in time. By exploiting the temporal correlations of sensor data, we make a prediction model for the future query result. Using the predicted future query result, sensor nodes can effectively filter out unnecessary transmissions. The contributions of this paper are as follows:

(i) We propose a lossless sensor data compression method to reduce the size of data transmitted. Prime numbers are used to encode a value of each sensing attribute. Sensor data are represented as one integer by multiplying encoded prime numbers.
(ii) In general, sensor data exhibit strong temporal correlations [5]. It means that the sensor data are quite similar during a short period of time and so future values can be predicted based on the previous measurements. Based on this fact, we devise a filtering-based data collection method using the previous query result.
(iii) The degree of temporal correlation varies according to sensor data. Therefore, we devise a model to capture the changing pattern of sensor data. Based on this model, the future query result is accurately predicted. Using the predicted query result, sensor nodes can filter out unnecessary transmissions.
(iv) Extensive experiments are conducted to evaluate the performance of the proposed approach. The results show that the proposed approach processes iceberg queries more energy efficiently than an existing approach while providing a highly accurate query result.

The remainder of this paper is organized as follows. Section 2 reviews the related work. In Section 3, we present the proposed approach for energy-efficient iceberg query processing. The experimental results are shown in Section 4. Finally, in Section 5, we conclude our work.

2. RELATED WORK

The term *iceberg query* was first introduced to describe queries that compute aggregate functions over an attribute (or a set of attributes) to find aggregate values above a certain threshold [1]. The reason why they call such queries iceberg queries is that the number of above-threshold data is often very small (the tip of an iceberg), relative to a large amount of input data (the iceberg). Traditionally, iceberg queries are commonly used in many applications, including data warehousing, information retrieval, market basket analysis in data mining, clustering and copy detection. Fang *et al.* [1] propose efficient iceberg query processing algorithms for the traditional databases in terms of the memory usage and the disk accesses.

Recently, iceberg query processing over stream databases has attracted much research effort. For example, iceberg queries can be used to detect distributed denial of service attacks or discover worms and other anomalies in network monitoring applications. Manjhi *et al.* [6] study the problem of processing iceberg queries in the union of multiple distributed data streams. They arrange nodes in a multi-level communication structure. Each node uses a synopsis data structure to find local icebergs, and then sends the synopsis to its parent in the communication structure. Different error tolerances are assigned to nodes according to their level in the hierarchy. The approximate synopses are passed from leaves to the root and combined incrementally within the corresponding error tolerances. However, this approach assumes that a globally frequent item is also frequent locally somewhere. If data are finely distributed across all nodes so that they do not appear much at any one location, this approach is not appropriate.

Zhao *et al.* [7] propose algorithms to effectively process iceberg queries without assuming that a globally frequent item is also locally frequent. They propose a sampling-based scheme and a counting-sketch-based scheme. In the sampling-based scheme, each node samples a list of data items along with their frequency counts and sends them to the server. In the counting-sketch-based scheme, each node summarizes its data into a counting sketch and samples a small percentage of identities of the items and sends both to the server. However, this approach is based on the flat infrastructure.

Zhao *et al.* [8] introduce the idea of using summable sketches for global iceberg query processing. They sum the sketches from individual nodes to get an aggregate sketch. Owing to the nature of summable sketches, it does not matter how the items are distributed along the nodes or in what order the data are aggregated. Also, it can be applied to any topology structure. However, they do not consider the energy saving aspect.

In a sensor network environment, sensor nodes have very limited battery resources. Therefore, minimizing the energy consumption is the most important issue in sensor networks. One main strategy for reducing the energy consumption is to do some query processing tasks in the network, as suggested by Sharaf *et al.* [9] and Yao and Gehrke [10]. The in-network aggregation reduces overall energy consumption by performing the computation within the network and reducing the amount of transmissions to the base station. Madden *et al.* [3] propose a generic aggregation service called TAG for *ad hoc* networks of TinyOS motes. They classify the types of aggregates supported by the system, focusing on the characteristics of aggregates that impact their performance and fault tolerance. Aggregates are processed in network by computing over the data as they flow through the sensors.

A second main strategy for conserving energy is using approximation. Deshpande *et al.* [11] propose a model-driven data acquisition technique. Rather than directly querying the sensor network, they build a model from stored and current sensor data, and answer queries by consulting the model. Sensor nodes are used to acquire data only when the model itself is not sufficiently rich to answer the query with acceptable confidence. Therefore, the energy consumption of sensor nodes can be reduced. Chu *et al.* [2] also propose an approximate data collection method using probabilistic models. The basic idea is to maintain a pair of dynamic probabilistic models over the sensing attributes, with one copy distributed in the sensor network and the other at the base station. At every time instance, the base station simply computes the expected values of the sensing attributes according to the model and uses it as the query answer. This requires no communication. However, these modeling approaches require an expensive learning phase to make an appropriate model. If the constructed model does not reflect the current data distribution, the learning phase should be re-applied. Furthermore, these approaches are mainly focused on processing the selection-type queries.

There also has been some work that considers more complicated queries than selection. Shrivastava *et al.* [12] propose a data aggregation scheme for computing approximate quantiles such as median and histogram. They present a structure, q-digest, which sensor nodes used to summarize the data they receive into a fixed size message, with approximation bounds.

Modeling patterns and correlations in data, and using them to reduce the data transmission rate has been a central theme in the data compression literature [13]. By exploiting some correlations in sensor data, the communication cost can be reduced. Guestrin *et al.* [14] propose a distributed regression for in-network modeling of sensor data. Based on the kernel linear regression, they model the entire sensor data from all sensor nodes. Rather than transmitting sensor data, sensor nodes transmit constraints on the model parameters. Therefore, it can predict the behavior of sensor data with a minimal communication cost. Deligiannakis *et al.* [15] propose a data compression technique, designed for historical data collected in sensor networks. They split the historical data into intervals of variable length and encode each of them using the base signal. The values of the base signal are extracted from the real measurements and maintained dynamically as data change.

In networking areas, there has been much research on energy-efficient routing for sensor network environments. Sensor

protocols for information via negotiation [16] and Directed diffusion [17] are data-centric routing protocols which find routes from multiple sources to a single destination that allows in-network aggregation of redundant data. Low energy adaptive clustering hierarchy [18], Hybrid energy-efficient distributed clustering [19] and GEodesic sensor clustering (GESC) [20] are clustering-based routing protocols. In the clustering-based routing protocol, sensor nodes are grouped into clusters and a cluster head is selected from each cluster. The cluster head aggregates sensor data from its member nodes and transmits the aggregated sensor data to the base station through the other cluster heads. Any energy-efficient routing protocol can be used to construct the routing topology in our proposed approach.

3. PROPOSED APPROACH

The goal of our approach is to provide an energy-efficient iceberg query processing technique in sensor networks. A formal definition of the problem is given in Section 3.1. An overview of our proposed approach is described in Section 3.2. Section 3.3 provides a lossless sensor data compression method. A filtering-based data collection method is described in Section 3.4.

3.1. Problem definition

This study considers a wireless sensor network that consists of the base station and n sensor nodes. It is assumed that the base station has a continuous power supply. In contrast, the sensor nodes are powered by batteries. Each sensor node is equipped with k types of sensors (i.e. temperature, humidity, soil moisture, etc.). A sensor node generates sensor data ($nodeID$, $v_1, v_2, \ldots, v_k$, $timestamp$), where

(i) $nodeID$ is the sensor node identifier;
(ii) v_i is the sensing value measured by the sensor i, where $1 \leq i \leq k$. It can be considered a value of an attribute in relational databases;
(iii) $timestamp$ is the time of measuring sensing values.

The range of each sensing value is set to the minimum and maximum values that can be measured by the corresponding sensor. It is assumed that sensing values are integers.

Since the communication range of a sensor node is limited to its local area, only close sensor nodes can directly communicate with each other. Therefore, a routing tree is used to transmit sensor data to the base station. It is assumed that there is no transmission failure. By retransmission, the transmission failure can be solved.

Based on this sensor network environment, we devise a best-effort approach that reduces the energy consumption of the sensor nodes without undue sacrifice in result quality. The prototypical iceberg query that we consider in this paper is to find a set of distinct sensor data whose total counts across all sensor nodes exceed a certain threshold T. The following is an SQL form of this query:

```
SELECT      A_1, A_2, ..., A_k, count(*)
FROM        Sensors
GROUP BY    A_1, A_2, ..., A_k
HAVING      count(*) ≥ T
SAMPLE PERIOD x
FOR d
```

The `SELECT` clause specifies attributes and aggregates from sensor data: A_i, where $1 \leq i \leq k$ is an attribute that represents a sensor type. COUNT is used as an aggregate function. *Sensors* represents a logical table that has sensor data generated from all sensor nodes. The `GROUP BY` clause classifies sensor data into different groups according to the specified attributes. The `HAVING` clause eliminates groups that do not satisfy the specified condition. `SAMPLE PERIOD` and `FOR` clauses specify the rate of query answers and the life time of the query, respectively.

Each sensor node transmits data to the base station that consists of a set of attributes specified in the `SELECT` clause and its aggregate value. It is assumed that the size of an attribute is 64 bits and the size of the aggregate value is 32 bits because COUNT is used as the aggregate.

The query results are maintained by the base station and provided to users by the base station.

3.2. Overall approach

In this section, we provide an overview of our proposed approach. Figure 1 summarizes this through an example.

To conserve energy of sensor nodes, we devise a lossless sensor data compression method and a filtering-based data collection method. In Fig. 1, query processing procedure at time $t = 3$ is described. Initially, the base station collects the sensor data from all sensor nodes until it has two previous query results for time $t = 1$ and $t = 2$. A routing tree is used for transmissions. Basically, an in-network aggregation is performed to reduce the number of data transmitted. For the next sampling instance $t = 3$, the base station predicts the possible query result for this time using the previous query results at $t = 1$ and $t = 2$. Since sensor nodes are distributed data sources, sensor data from all sensor nodes should be transmitted to the base station to find out the final aggregate values of them. However, most of these transmissions are not the query answer. Therefore, transmitting data, which are not the query answer, wastes valuable energy of sensor nodes. To find out the unnecessary transmissions within the network and suppress these transmissions, we exploit the previous query results. Since sensor data exhibit strong temporal correlations, we can infer the next query result from the past one. The aggregate values of all possible tuples are estimated using our prediction model. A tuple consists of a set of attributes specified in the `SELECT` clause. Tuples that have larger estimated aggregate values than

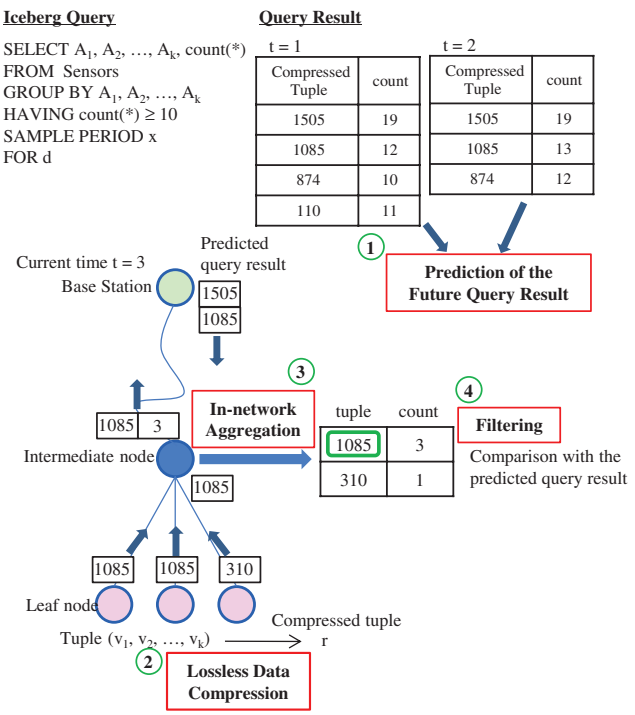

FIGURE 1. Overview of the proposed approach.

the threshold are considered as the possible query result at $t = 3$. The base station transmits this possible query result to the sensor nodes that perform in-network aggregations (i.e. sensor nodes except leaf nodes). Since leaf nodes only have their own sensor data, they do not perform in-network aggregations. Therefore, the possible query result is only sent to intermediate sensor nodes. Starting from the leaf nodes, tuples are transmitted along the routing tree. The size of one tuple corresponds to the number of sensing attributes specified in the SELECT clause. Usually, the number of sensing attributes used in the iceberg query is large and so does the size of one tuple. Therefore, sensor nodes consume a lot of energy due to the basic size of a tuple. To reduce the size of one tuple, we devise a lossless sensor data compression method. By applying this compression method to tuples, each tuple can be represented as one integer. Sensor nodes transmit these compressed tuples to their parent nodes. The lossless sensor data compression method will be described in detail in Section 3.3.

Intermediate sensor nodes perform in-network aggregations based on the received tuples from their child nodes and its own tuple. The predicted query result is used in this step. Tuples that have smaller aggregate values than the threshold are compared with the predicted query result. Since tuples that already have larger aggregate values will be query answers, we do not consider them. For tuples that have smaller aggregate values than the threshold, sensor nodes filter out tuples that will not be included in the query result based on the knowledge of the possible query result. Therefore, our approach can greatly reduce the energy consumption of sensor nodes. The filtering-based data collection method will be presented in Section 3.4.

3.3. Lossless sensor data compression

In this section, we discuss the lossless sensor data compression method to represent data to be transmitted compactly and energy efficiently. Radio communication dominates sensors' energy consumption. The amount of energy spent on communication of data with x bytes is given by $\sigma + \delta x$, where σ and δ represent the per-message and per-byte costs, respectively [21]. Therefore, the energy consumption of sensor nodes depends on the size and the number of data transmitted. To process the iceberg query, sensor nodes have to transmit data that consists of k sensing attributes (i.e. $(v_1, v_2, \ldots, v_k)$) at a fixed sampling rate. We call these data tuples. Usually, the number of attributes specified in the iceberg query is large and so is the tuple to be transmitted. Thus, it cannot help consuming a large amount of energy due to the basic size of one tuple. Since sensor nodes have very limited energy resources, an effective data size reduction method is required to save the energy consumption.

There have been many compression techniques proposed to support lossless compression in sensor networks. Among the techniques, S-LZW [22] was specifically devised for sensor networks. S-LZW is an adapted version of the Lempel–Ziv–Welch (LZW) [23] designed for resource-constrained sensor nodes. S-LZW uses adaptive dictionary techniques with dynamic code length. However, S-LZW suffers from the growing dictionary problem because each new sensor datum produces a new entry in the dictionary. For iceberg queries, the number of distinct tuples is very large, and thus the dictionary size becomes problematic. Lossless entropy compression (LEC) [24], which is based on static Huffman encoding, exploits the temporal correlation of sensor data using a fixed dictionary. Since the dictionary size is fixed, LEC does not suffer from the growing dictionary problem. However, LEC requires prior knowledge of the statistical characteristics of sensor data and thus the communication overhead is high. Therefore, these approaches are not suitable for our considered environment.

To reduce the size of one tuple, we use the prime numbers based on the *Fundamental Theorem of Arithmetic* in the number theory. There have been some uses of the unique property of prime numbers in other researches such as XML [25] and radio frequency identification data management [26]. The property was used for an effective encoding of data for query processing in these researches; however, we use the property for data compression.

THEOREM 3.1 (Fundamental theorem of arithmetic). *Every integer $n > 1$ can be represented as a product of prime factors in only one way, apart from the order of the factors.*

For example, the prime factors of 330 are 2, 3, 5 and 11 (i.e. $330 = 2 \times 3 \times 5 \times 11$). There is no other possible set of prime numbers that can be multiplied to make 330. Based on this theorem, we devise the lossless sensor data compression method. Using the unique property of prime numbers, a tuple is efficiently compressed into one integer without introducing the growing dictionary problem and the communication overhead for obtaining the statistical characteristics of sensor data.

Assume that v_i has a value between min_{v_i} and max_{v_i} (i.e. $v_i \in [min_{v_i}, max_{v_i}]$), where min_{v_i} represents the minimum value of the sensing attribute A_i, max_{v_i} represents the maximum value of it, and $1 \leq i \leq k$ (k = # of sensing attributes). Let $D(A_i)$ be an increasing ordered set of all sensing values for A_i (i.e. $D(A_i) = \{v_{i1}, v_{i2}, \ldots\}$, where v_{i1} and v_{i2} denote the first and the second elements of $D(A_i)$, respectively). To make an encoding table, the prime numbers in increasing order starting with 2 and ending with the pth prime number are assigned to all elements in $D(A_i)$, where p is the sum of the cardinalities of all $D(A_i)$ (i.e. $p = \sum_{i=1}^{k} |D(A_i)|$). Let $p_1 = 2 < p_2 = 3 < p_3 = 5 < \cdots$ be the prime numbers (in increasing order). Starting from v_{i1} of all $D(A_i)$, the prime numbers from p_1 to p_k are assigned to each v_{i1} until all $v_{il} \in D(A_i)$ are processed, where l represents the cardinality of $D(A_i)$ that might be different between A_is depending on the range of each A_i ($v_{11} \Rightarrow 2$, $v_{21} \Rightarrow 3, \ldots, v_{k1} \Rightarrow p_k$, $v_{12} \Rightarrow p_{k+1}$, $v_{22} \Rightarrow p_{k+2}, \ldots v_{kl} \Rightarrow p_p$).

Figure 2 shows an example of the encoding table. Consider the case where each sensor node is equipped with temperature, humidity and soil moisture sensors. The ranges for sensing values of temperature, humidity and soil moisture are [1, 5], [1, 5] and [3, 6], respectively. This is used as a running example throughout the paper. The number of all possible sensing values is 14. Therefore, 14 smallest prime numbers are assigned to sensing values starting from the minimum values to the maximum values of them. The temperature value 1 is assigned to 2, the humidity 1 is assigned to 3 and the soil moisture 3 is assigned to 5, etc.

Based on the encoding table, a sensing value v_i is encoded to its corresponding prime number ($Prime(v_i)$). Let a tuple be $(v_1, v_2, \ldots, v_k)$. By encoding all sensing values to their corresponding prime numbers, the tuple is changed to $(Prime(v_1), Prime(v_2), \ldots, Prime(v_k))$. These encoded prime numbers are multiplied to make one integer. Therefore, we can compactly represent an original tuple with only one integer $((v_1, v_2, \ldots, v_k) \rightarrow P = (Prime(v_1) \times Prime(v_2) \times \cdots \times Prime(v_k)))$. This one integer P can be uniquely decomposed into prime numbers by Theorem 3.1. If a tuple consists of k sensing attributes, where $k \geq 1$, the data compression ratio is $1/k$; k is the number of attributes specified in the iceberg query and this number is usually not small. As k increases, we can get a higher compression ratio. In the above example, let a tuple be (2, 4, 4). Then, temperature value 2, humidity value 4 and soil moisture value 4 are encoded to 7, 31 and 13, respectively ($(2, 4, 4) \rightarrow (7, 31, 13)$). By multiplying encoded prime numbers together, the original tuple (2, 4, 4) can be represented as one integer 2821 ($2821 = 7 \times 31 \times 13$). The original sensor values are easily reconstructed by finding prime factors of 2821. Therefore, we can reduce the size of a tuple without any loss of data and save the energy consumption of sensor nodes.

For sensing values that are real numbers, the proposed compression method can be applied by scaling up the original sensing value by a factor of powers of 10. For example, a sensing value v_i is in the form of $v_i = a_0.a_1a_2a_3\ldots$, where a_0 is the integer part of v_i, and $a_1, a_2, a_3\ldots$ are the digits forming the fractional part of v_i. According to the digits of the fractional part, the scaling factor is determined. If the number of the digits of the fractional part is 2, 10^2 is multiplied to the original sensing value and making it an integer. Then, the same procedure is applied to compress sensor data.

If the range of values of a sensing attribute is large, many prime numbers are required to make the encoding table. Assigning prime numbers sequentially to sensing values within domain ranges of sensing attributes may increase the value of the compression result (an integer generated by multiplying corresponding prime numbers of sensing values) and cause an overflow. To prevent this problem, history information about frequencies of sensing values is used. The sensing values of each sensing attribute are sorted by the frequency in descending order. Based on the sorted sensing values of each sensing attribute, prime numbers are assigned sequentially. In the example of Fig. 2, let the sorted sensing values of temperature, humidity and soil moisture be (2, 5, 1, 3, 4), (2, 4, 1, 3, 5) and (4, 3, 5, 6), respectively. Prime number 2 is assigned to

temperature $\in [1, 5]$ humidity $\in [1, 5]$ soil moisture $\in [3, 6]$

temperature	1	2	2	7	3	17	4	29	5	41
humidity	1	3	2	11	3	19	4	31	5	43
soil moisture	3	5	4	13	5	23	6	37		

sensing value prime number

FIGURE 2. An example of an encoding table.

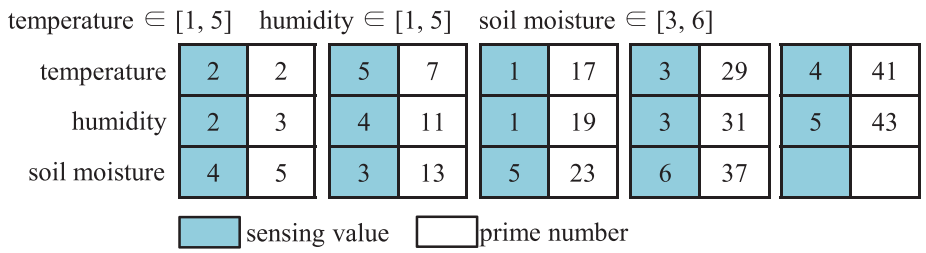

FIGURE 3. An example of an encoding table based on the frequency information.

temperature 2, prime number 3 is assigned to humidity 2 and prime number 5 is assigned to soil moisture 4. Figure 3 shows the encoding table based on the history information about frequencies of sensing values. The tuple (2, 4, 4) is compressed to one integer 110 ($110 = 2 \times 11 \times 5$). Therefore, the tuple is compressed to a smaller integer than that of the encoding table in Fig. 2 ($2821 = 7 \times 31 \times 13$).

There are a few cases where the assigned prime numbers by using the encoding table based on the frequency information are larger than those by using the encoding table which is made by assigning prime numbers sequentially to sensing values. However, the use of the frequency information results in assigning small numbers in most cases because frequently generated sensing values are represented as small integers. Since sensing values typically follow similar patterns, the proposed scheme is beneficial.

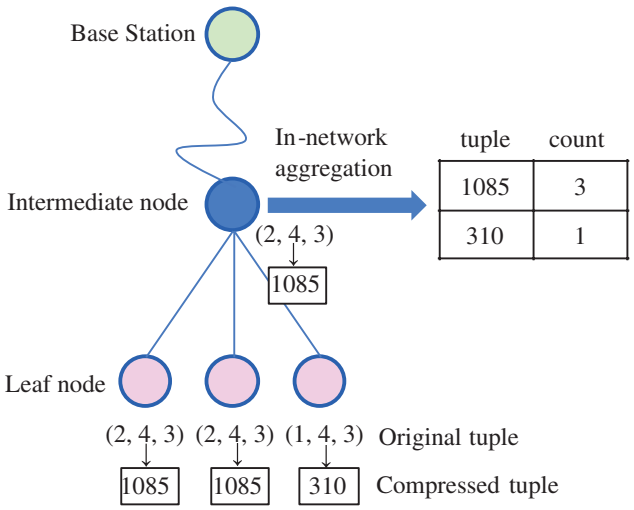

FIGURE 4. An example of the in-network aggregation.

3.4. Filtering-based data collection

The energy consumption of sensor nodes can be minimized by reducing the number of data transmitted. The result of the iceberg query is a set of tuples whose aggregate values exceed a certain threshold. The transmissions of tuples whose final aggregate values are smaller than the threshold waste valuable energy of sensor nodes. However, the final aggregate value of a certain tuple is determined by how many tuples generated from other sensor nodes are same as the considered tuple. Since sensor nodes are distributed, all sensor nodes should transmit their tuples to the base station to determine the final aggregate values of distinct tuples. If the possibility that a tuple belongs to the query result can be known in advance, sensor nodes can suppress transmissions of tuples that will not be query answers. Thus, the energy consumption of sensor nodes can be greatly reduced. We devise a filtering-based data collection method based on the model that predicts the possible future query result.

Basically, an in-network aggregation is performed to reduce the number of data transmitted. Figure 4 shows an example of the in-network aggregation. Each leaf node in the routing tree transmits its own compressed tuple to the parent node and the parent node applies the in-network aggregation based on both the received and its own tuples. Tuples are grouped according to the values of sensing attributes specified in the `GROUP BY` clause and then the aggregate function (COUNT) is applied to the tuples in each group.

The final result consists of a set of tuples that satisfy the `HAVING` condition ($count(*) \geq T$). Although the in-network aggregation can reduce the energy consumption, only performing this is not helpful to determine whether a certain tuple will be a query answer or not. If the possibility that a certain tuple will be in the query result can be known in advance, we can use this information to avoid unnecessary transmissions. In iceberg query processing, an unnecessary transmission is a transmission of the tuple that has the final aggregate value smaller than the threshold.

To suppress unnecessary transmissions, topology information is first used. Each sensor node except leaf nodes has its intermediate result denoted by $IR = \{(r_i^c, count(r_i^c)) | 1 \leq i \leq m, m$ *is the number of distinct tuples,* $r_i^c = Prime(v_1) \times Prime(v_2) \times \cdots \times Prime(v_k)$*, where* $Prime(v_i)$ *is the prime number for the sensing value* v_i*, and* $count(r_i^c)$ *is a count value of* $r_i^c\}$. Based

on the aggregated values of r_i^c ($count(r_i^c)$), we obtain the minimum and the maximum aggregate values of them. These minimum and maximum aggregate values are classified into three categories as follows:

(i) Case 1 $min(count(r_i^c)) \geq T$;
(ii) Case 2 $min(count(r_i^c)) < T$ and $max(count(r_i^c)) \geq T$;
(iii) Case 3 $max(count(r_i^c)) < T$.

If IR satisfies the condition of Case 1, all tuples in IR have to be transmitted to the base station because these are the query answers. Otherwise (Cases 2 and 3), the topology information is used to determine whether a tuple will be included in the final result or not. For a tuple r_i^c that has a smaller $count(r_i^c)$ than the threshold T, the difference $diff_i = T - count(r_i^c)$ is calculated. Since tuples are transmitted along the routing tree, a sensor node knows all tuples from its descendant sensor nodes. However, the sensor node cannot know tuples generated by (1) its ancestor nodes, (2) subtrees of its sibling nodes and (3) ancestor nodes of the sibling nodes. In other words, the sensor node only knows tuples from its descendant nodes and itself, and others are unknown. Let s be the number of descendant sensor nodes of a sensor node. The number of unknown sensor nodes of it, u, is obtained by $u = n-(s+1)$, where n is the total number of sensor nodes. For r_i^c, if $diff_i + u$ is smaller than T, $count(r_i^c)$ cannot increase by more than T. That is, although the aggregation of all tuples from unknown sensor nodes are finished, $count(r_i^c)$ is still smaller than T. Therefore, r_i^c cannot be the query answer. If a tuple satisfies the condition $diff_i + u < T$, we do not need to transmit it. For a tuple that satisfies the condition $diff_i + u \geq T$, after finishing the aggregation of all unknown tuples, this tuple may be included in the final result or not. In this step, sensor nodes cannot be sure whether the tuple is the query answer or not. Therefore, we use the previous query result for tuples that satisfy the condition $diff_i + u \geq T$. Figure 5 illustrates this.

In general, there are significant temporal correlations in sensor data. Sensor data have quite similar sensing values in a short period of time. Therefore, the future values can be predicted based on the previous measurements. For iceberg query processing, we can consider that tuples in the previous query result will not be changed significantly due to the temporal correlation of sensor data. The temporal correlation of sensor data can help us in estimating the next query result from the previous one.

The basic process of using the previous query result is as follows:

(1) The base station estimates the future count values of tuples based on the previous query result. The detailed explanation of how to estimate the future count values will be given later.
(2) Tuples that have larger estimated count values are considered as the possible future query result. These tuples are transmitted to sensor nodes except leaf nodes.
(3) A sensor node that receives the possible future query result from the base station compares the received data with its intermediate result to decide which tuples should be filtered out.
   (a) If a tuple in an intermediate result is equal to any tuple in the possible future query result, the sensor node transmits it to the parent node.
   (b) Otherwise, the tuple is not transmitted.

Figure 6 shows an example of this process. Assume that the threshold T is 10 and all tuples in the intermediate result satisfy the condition $diff_i + u \geq T$. The query result R consists of a set of data denoted by $(r_i^c, count(r_i^c))$, where $r_i^c = Prime(v_1) \times Prime(v_2) \times \cdots \times Prime(v_k)$ (i.e. r_i^c is a compressed representation of the original tuple $r_i = (v_1, v_2, \ldots, v_k)$) and $count(r_i^c)$ is a count value of r_i^c. Let R be {(1505, 20), (1085, 15), (874, 12)}. The base station estimates the future count value of each tuple based on R. If the estimated result is {(1505, 19), (1085, 13), (874, 8)}, then 1505 and 1085 are transmitted to sensor nodes because their count values are larger than 10. A sensor node compares its intermediate result {(1085, 3), (310, 1)} with the received data 1505 and 1085. Although the count value of 1085 is smaller than 10, it is transmitted to the parent node since it is equal to the received data. In the case of 310, the sensor node does not transmit it because it is not matched with any received data. Therefore, we can reduce the number of transmitted data from 2 to 1.

The number of data to be transmitted can be reduced by exploiting the temporal correlation of sensor data. However, the degree of the temporal correlation varies according to sensor data. Therefore, we need a well-defined model to capture the temporal correlation in sensor data. To represent the degree of the temporal correlation of sensor data, we consider both the age and the changing pattern of the query result. If the timestamp of the previous query result is a recent one compared with the current time, we can highly trust this result. This can be

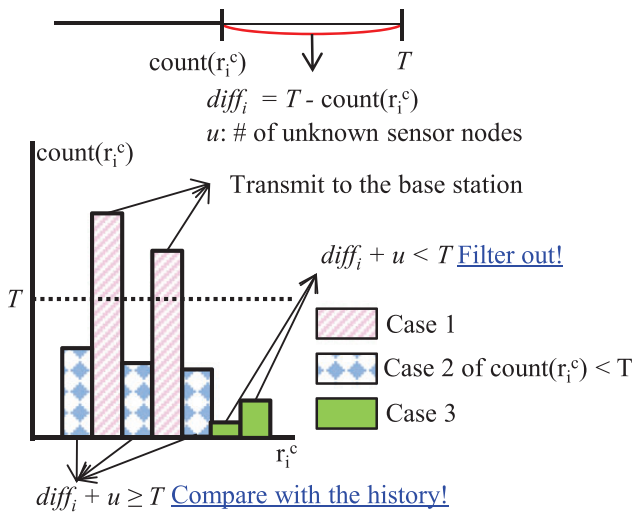

FIGURE 5. Possible cases for counts in the intermediate result.

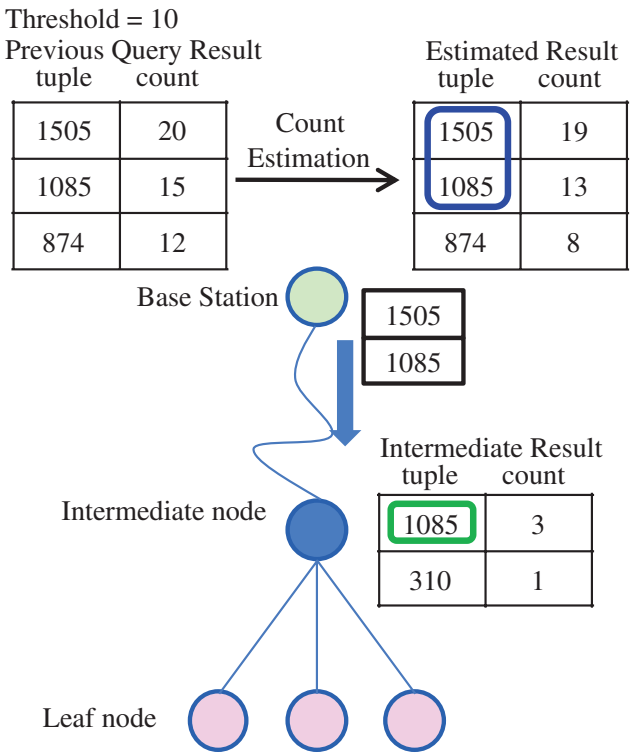

FIGURE 6. An example of filtering-based data collection.

formalized via a *time-decaying function* that assigns a weight to the query result based on the age of it. The age of the previous query result is $a = t_{\text{current}} - t_{\text{previous}}$, where t_{current} and t_{previous} represent the current time and the previous time, respectively.

DEFINITION 3.1. *A function $f(a)$ is a time-decaying function if it satisfies the following properties*:

(1) $f(0) = 1$ *and* $0 \leq f(a) \leq 1$ *for all* $a \geq 0$;
(2) f *is monotone decreasing*: *if* $a_1 \geq a_2$, *then* $f(a_1) \leq f(a_2)$.

Some popular decay functions are as follows:

(i) *Sliding window*: For a window of size W, the function is defined by $f(a) = 1$ for $a < W$ and $f(a) = 0$ for $a \geq W$.
(ii) *Exponential decay*: The exponential decay is defined by $f(a) = e^{-\lambda a}$ for $\lambda > 0$.
(iii) *Polynomial decay*: The polynomial decay is defined by $f(a) = (a+1)^{-\alpha}$ for some $\alpha > 0$.

The graphs of these decay functions are given in Fig. 7. Sliding window only considers data within the recent window of size W. All data within the window have equal weights (i.e. 1), while the other data not in this window have zero weights. For data outside the window, sliding window cannot finely adjust their weights according to the ages. Exponential decay can solve the

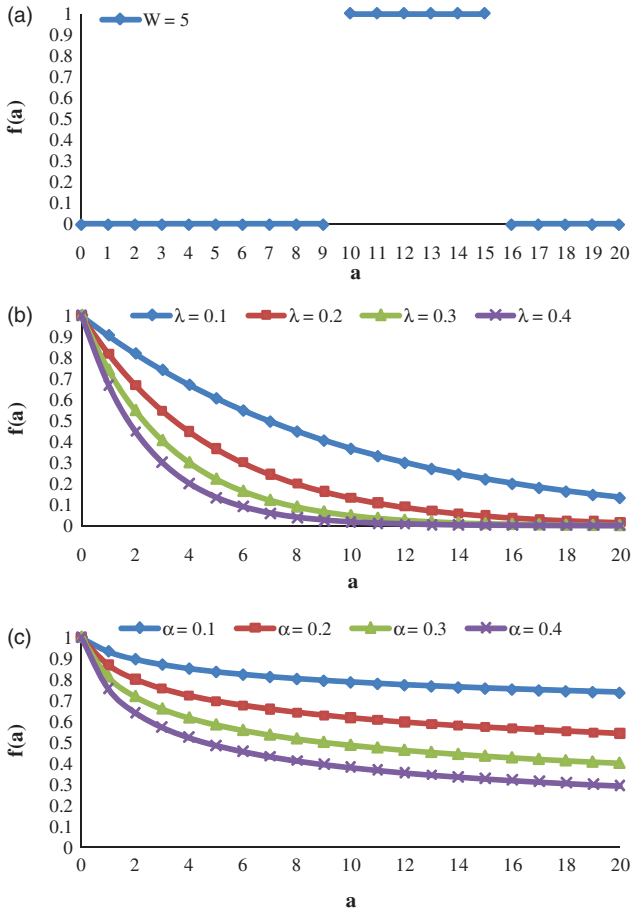

FIGURE 7. Decay functions. (**a**) Sliding window, (**b**) exponential decay and (**c**) polynomial decay.

problem of sliding window by differentiating the rate of change according to the age of data. However, it has an extremely fast decreasing rate. Since sensor data will not be changed severely for consecutive times due to the temporal correlation, it is not suitable for sensor data. Polynomial decay has a smooth changing rate compared with exponential decay. It can properly tune the decreasing rate of weight with the age. Therefore, we use a polynomial decay as the time-decaying function. By using the time-decaying function, we can effectively represent the reliability of the previous query result.

To find out how many same tuples are continuously included in the query results, we measure the similarity between the previous query results. If there exist many same tuples within the two consecutive previous query results, we can consider that the degree of temporal correlation of sensor data is very high. To represent the similarity of two previous query results quantitatively, *Cosine Similarity* is used. Cosine similarity measures the similarity by finding the cosine of the angle between two vectors of d dimensions.

Given two closest previous query results $R_{t-\Delta}$ and $R_{t-(\Delta+\phi)}$ of the current time t, where $t-\Delta$ and $t-(\Delta+\phi)$ denote the

closest time from t and from $t-\Delta$, respectively, the base station performs the following procedures to measure the similarity between them:

(1) Tuples in $R_{t-\Delta}$ and $R_{t-(\Delta+\phi)}$ are decompressed to restore the original tuples. The encoded prime numbers for values of sensing attributes are obtained by the prime factorization of the compressed tuple. Based on the encoding table, these prime numbers are decoded as their original sensing values.

(2) For decompressed values of each sensing attribute A_i at time $t-\Delta$ and $t-(\Delta+\phi)$, two bags, $B_{t-\Delta}(A_i)$ and $B_{t-(\Delta+\phi)}(A_i)$, are made. A bag is represented by a set of sensing values and their count values. The element of the bag is denoted by $\langle v, c\rangle$, where v is a sensing value and c is its count. The elements of the bag are sorted by v in ascending order (i.e. $B_t(A_i) = \{\langle v_{i1}, count(v_{i1})\rangle, \langle v_{i2}, count(v_{i2})\rangle, \ldots\}$, where $v_{i1} < v_{i2} < \cdots$). If the values of the decompressed one are same, the count value of them are summed up.

(3) The similarity between $B_{t-\Delta}(A_i)$ and $B_{t-(\Delta+\phi)}(A_i)$ are calculated:

(a) The frequency vectors are made to represent the count of each value of the sensing attribute. Extract the count value of each sensing value from $B_{t-\Delta}(A_i)$ and $B_{t-(\Delta+\phi)}(A_i)$ and make their frequency vectors $F_{t-\Delta}(A_i)$ and $F_{t-(\Delta+\phi)}(A_i)$. The dimension of the frequency vector is the sum of the number of the distinct sensing values in $B_{t-\Delta}(A_i)$ and $B_{t-(\Delta+\phi)}(A_i)$. Starting from the first dimension to the final dimension of the frequency vector, each dimension corresponds to the sensing values of the bags (i.e. first dimension $\Rightarrow v_{i1}$, second dimension $\Rightarrow v_{i2}, \ldots$ (in increasing order)).

(b) Calculate the cosine similarity between $F_{t-\Delta}(A_i)$ and $F_{t-(\Delta+\phi)}(A_i)$:

$$similarity(F_{t-\Delta}(A_i), F_{t-(\Delta+\phi)}(A_i)) = \frac{F_{t-\Delta}(A_i)\cdot F_{t-(\Delta+\phi)}(A_i)}{|F_{t-\Delta}(A_i)||F_{t-(\Delta+\phi)}(A_i)|}.$$

If two consecutive previous query results have many similar data, *similarity* has a value close to 1. Figure 8 shows an example of the similarity calculation. Assume that the current time is t and the base station has the query results at time $t-1$ and $t-2$. The base station decompresses the tuples of the query results. A_1, A_2 and A_3 denote sensing attributes such as temperature, humidity and soil moisture. Bags are made to represent the counts of values of the sensing attribute for time $t-1$ and $t-2$. For the sensing attribute A_1 in the decompressed result at $t-1$, the sensing value 1 exists in two tuples and their counts are 20 and 15. Thus, the total count of 1 is 35. Likewise, the counts for the other sensing values 2 and 3 are calculated by summing up the counts of the corresponding values. The bags for the sensing attribute A_1 are $B_{t-1}(A_1) = \{\langle 1, 35\rangle, \langle 2, 13\rangle, \langle 3, 30\rangle\}$ and $B_{t-2}(A_1) = \{\langle 1, 34\rangle, \langle 2, 10\rangle, \langle 3, 33\rangle, \langle 5, 20\rangle\}$. Since the number of distinct sensing values in $B_{t-1}(A_1)$ and $B_{t-2}(A_1)$ is 4 (1, 2, 3, 5), the dimension of the frequency vector becomes 4. The first, second, third and fourth dimensions of the frequency vector represent the counts of the sensing values 1, 2, 3 and 5, respectively. The frequency vectors for time $t-1$ and $t-2$ are $F_{t-1}(A_1) = (35, 13, 30, 0)$ and $F_{t-2}(A_1) = (34, 10, 33, 20)$.

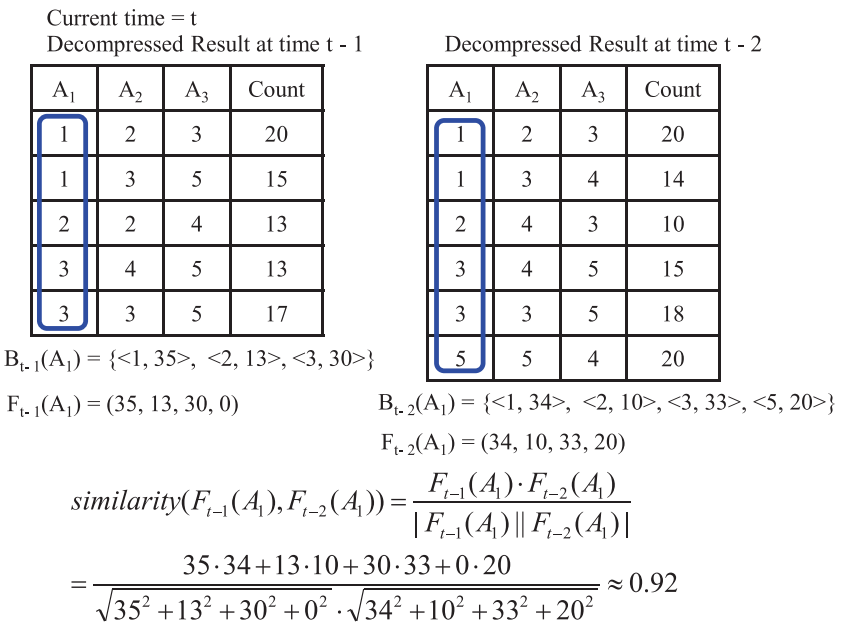

FIGURE 8. An example of the similarity calculation.

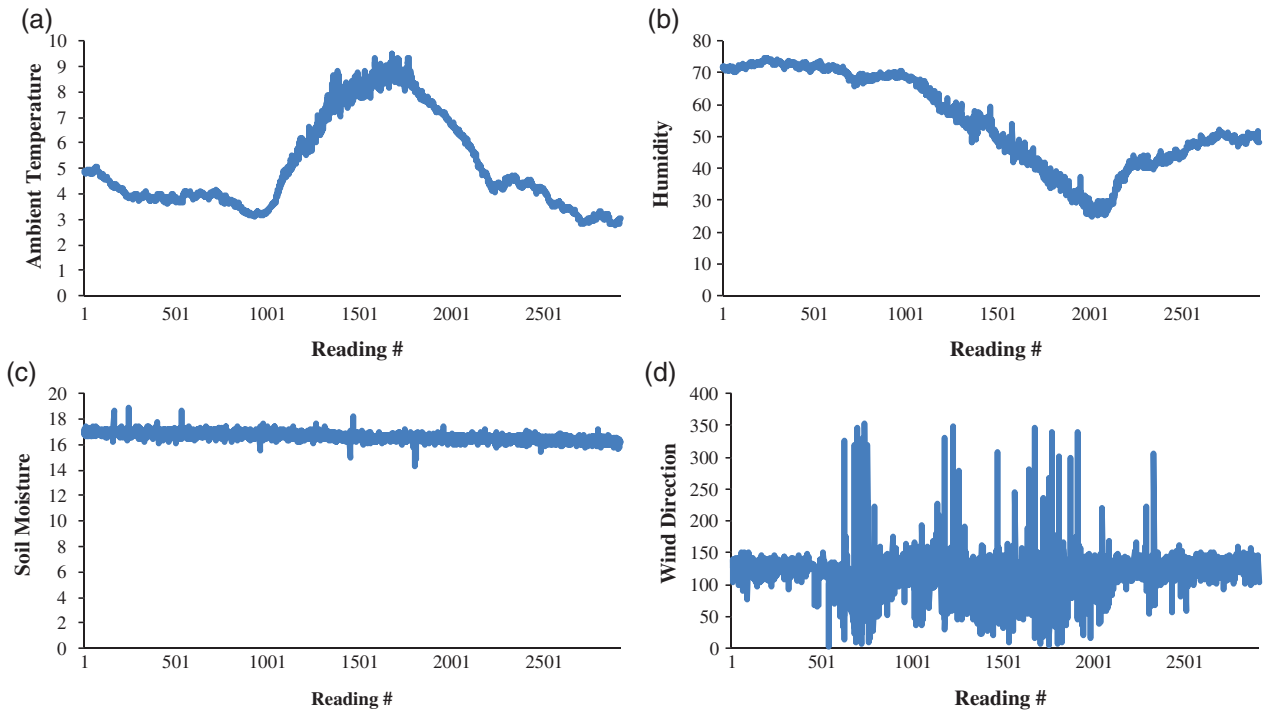

FIGURE 9. Traces of sensing attributes in the LUCE data set. (**a**) Ambient temperature, (**b**) humidity, (**c**) soil moisture and (**d**) wind direction.

Based on $F_{t-1}(A_1)$ and $F_{t-2}(A_1)$, the cosine similarity is calculated as shown in Fig. 8.

Using the age and the temporal correlation of the query result, the prediction model for estimating the future count of a tuple is devised. The probability that a tuple $r = (v_1, v_2, \ldots, v_k)$ will be changed to $r' = (v'_1, v'_2, \ldots, v'_k)$ is computed by

$$p(r \rightarrow r') = \prod_{i=1}^{k} p(v_i \rightarrow v'_i).$$

We assume that each sensing attribute is independent. Figure 9 shows the real traces (LUCE data set[1]) of ambient temperature, humidity, soil moisture and wind direction data over a period of one day from a single sensor node. As shown in Fig. 9, we cannot find any correlation between different sensing attributes. This justifies the assumption of independence between sensing attributes. Based on the *multiplication rule* for independent events, $p(r \rightarrow r')$ is calculated by multiplying each probability that a sensing value v_i will be changed to v'_i, which is denoted as $p(v_i \rightarrow v'_i)$. We calculate $p(v_i \rightarrow v'_i)$ as follows:

(1) If $v_i = v'_i$:

$$p(v_i \rightarrow v_i) = f(a)similarity(F_{t-\Delta}(A_i), F_{t-(\Delta+\phi)}(A_i)),$$

where a is the age of the previous query result at time $t - \Delta$.

(2) Otherwise

$$p(v_i \rightarrow v'_i) = (1-p(v_i \rightarrow v_i)) \frac{(1 - p(v_i \rightarrow v_i))/(|Dom(A_i)| - 1)|v_i - v'_i|}{\sum_{i=1}^{|Dom(A_i)|-1}(1 - p(v_i \rightarrow v_i))/(|Dom(A_i)| - 1)|v_i - v'_i|},$$

where $|Dom(A_i)|$ is the domain size of the sensing attribute A_i.

The probability that a sensing value will remain the same value as the previous one, $p(v_i \rightarrow v_i)$, is calculated by multiplying the time-decaying function and the similarity between two previous query results. The time-decaying function $f(a)$ is used to assign a larger weight to a younger query result. $similarity(F_{t-\Delta}(A_i), F_{t-\Delta+\phi}(A_i))$ is used to measure the degree of the temporal correlation. If the two consecutive previous query results have many similar sensing values, the sensor data can be considered to have a very strong temporal correlation. Thus, a large portion of the previous query result will be generated later in time.

The probability that a sensing value will be changed from the previous one, $p(v_i \rightarrow v'_i)$, is calculated based on the probability $p(v_i \rightarrow v_i)$. We find that $1 - p(v_i \rightarrow v_i)$ is the probability that a sensing value v_i is changed to any v'_i, where

[1] http://sensorscope.epfl.ch/index.php/Environmental_Data.

$v_i' \in Dom(A_i) - \{v_i\}$. Because of the nature of the physical phenomenon, sensor data gradually increase or decrease in a short period of time. To reflect this, $|v_i - v_i'|$ is used in $p(v_i \to v_i')$. For example, let the domain range of the temperature be $D(temperature) = \{1, 2, 3, 4, 5\}$ and $p(1 \to 1) = 0.7$. Then, the probability that 1 is changed to any other value in $D(temperature) - \{1\}$ is 0.3. Since 2 will be generated with a higher probability than 5 in a short period of time, assigning uniform probability $(1 - p(1 \to 1))/(|Dom(temperature)| - 1)$ (i.e. 0.3/4) to each value in $D(temperature) - \{1\}$ is inappropriate. Therefore, we should give a different probability according to the value difference between v_i and v_i'. By dividing $(1 - p(1 \to 1))/(|Dom(temperature)| - 1)$ by the difference between 1 and the other value, we can give a higher probability for a value similar to 1 (i.e. $p(1 \to 2) = 0.3/(4 \times 1)$, $p(1 \to 3) = 0.3/(4 \times 2)$, $p(1 \to 4) = 0.3/(4 \times 3)$, $p(1 \to 5) = 0.3/(4 \times 4)$). We use $(1 - p(v_i \to v_i))/(|Dom(A_i)| - 1)|v_i - v_i'|$ in $p(v_i \to v_i')$ for this. One uses $(1 - p(v_i \to v_i))/\sum_{i=1}^{|Dom(A_i)|-1} (1 - p(v_i \to v_i))/(|Dom(A_i)| - 1)|v_i - v_i'|$ to make the sum of the probabilities $p(v_i \to v_i')$ equal to $1 - p(v_i \to v_i)$ (i.e. 0.3).

If sensing attributes are correlated with each other, the probability $p(r \to r')$ is calculated by the joint probability for the correlated with sensing attributes as follows:

$$p(r \to r') = p(v_1 \to v_1', v_2 \to v_2', \ldots, v_k \to v_k').$$

Based on the probability $p(r \to r')$, the count of a tuple r is estimated as follows:

$$count_{\text{estimated}}(r) = \sum_{r_i \in R_t} p(r_i \to r) count(r_i) + \left(\sum_{r_j \in U - R_t} p(r_j \to r) \right) \times \left(n - \sum_{r_i \in R_t} count(r_i) \right),$$

where r is a considered tuple, R_t is a previous query result at t, r_i is a tuple of R_t, $count(r_i)$ is a count value of r_i, U is a set of all possible tuples, r_j is a tuple of $U - R_t$ and n is the number of sensor nodes.

The set of all possible tuples U is obtained by permuting all values of the sensing attributes within their domain ranges. Since any tuple in U can be generated for the next time, all of them are considered to predict the future query result. We represent by $\sum_{r_i \in R_t} p(r_i \to r) count(r_i)$ how many tuples in R_t will be changed to r. For tuples that are in the previous query result, the actual counts of them are used for estimation. However, for tuples that are not in the previous query result, $U - R_t$, there is no way to know their actual counts because the base station does not have them. Therefore, we simply assume that sensor nodes that do not generate tuples in the previous query result can uniformly have tuples in $U - R_t$. By calculating $(\sum_{r_j \in U - R_t} p(r_j \to r))(n - \sum_{r_i \in R_t} count(r_i))$, we can reflect the possibility that tuples in $U - R_t$ will be changed to r.

The count values of all possible tuples are predicted by the equation of $count_{\text{estimated}}(r)$. Tuples that have larger counts than the threshold are transmitted to sensor nodes and used for suppressing the unnecessary transmissions. Therefore, we can greatly reduce the energy consumption of sensor nodes.

However, there is a case where the temporal correlation of sensor data is low. In this case, since sensor data change randomly with time, the similarity between the previous query results will be low. Therefore, we cannot accurately predict the next query result based on the previous query result. To deal with this case, prediction-based filtering is applied adaptively according to the similarity of two consecutive previous query results. If the similarity between the previous query results is below a certain threshold, only the topology information is used to filter out unnecessary transmissions. Otherwise, both the topology information and the predicted query result are used for filtering. Although this adaptive filtering increases the energy consumption of sensor nodes, it can provide more accurate query results.

4. EXPERIMENTAL EVALUATION

An experimental analysis was conducted to validate the proposed approach using our own simulator developed in Java. The simulator uses a simplified version of the time division multiple access protocol where sensor nodes communicate on different time slots in order to prevent collisions. The radio range of a sensor node is set to 150 m according to the specification of TinyNode.[2] The results demonstrate the energy efficiency and the accuracy of the proposed approach.

4.1. Experimental environment

Data sets and setting: Real-world and synthetic data sets are used to evaluate the proposed approach. For the real-world data set, the LUCE data set, that is a trace of measurements from 100 weather stations at the EPFL campus, is used. Each weather station (i.e. a sensor node) measures nine different sensing attributes: ambient temperature, surface temperature, solar radiation, relative humidity, soil moisture, watermark, rain meter, wind speed and wind direction. In the LUCE data set, we observe that the number of distinct tuples is very small, although the total number of tuples is very large. This corresponds to the characteristic of the iceberg query, that the size of the output data is extremely small compared with the size of the input data. If the number of sensing attributes is large, this tendency is intensified. Therefore, parts of sensing attributes are used to show the effect of various thresholds. The sensor nodes that have many unavailable data are not used in the experiments.

[2] http://www.tinynode.com/.

To show the effects of temporal correlations of sensor data, synthetic data sets are generated. One hundred sensor nodes are used and their locations are determined based on the positions of weather stations in the LUCE data set. Each sensor node generates values for nine sensing attributes. To control the temporal correlation of sensor data, the *tcd* parameter is used, which enables us to set the degree of changes in sensing values. Values for tcd range from 0 to 100. Based on the tcd value, sensor nodes that will generate the same sensing values as their previous ones are determined. The initial sensing values of sensor nodes are randomly generated in the range [0, 20]. *tcd*% of sensor nodes are randomly selected to assign the same sensing values as their previous ones. Sensing values of the other sensor nodes are randomly generated from the range [0, 20]. For example, if $tcd = 70\%$, then 70% of sensor nodes will generate the same sensing values as before and 30% of sensor nodes will generate different sensing values from their previous ones.

To test the scalability of the proposed approach, various sensor networks are constructed by increasing the number of sensor nodes. Sensor nodes are randomly placed in an area of $1000 \times 1000\,\mathrm{m}^2$ and we vary the number of sensor nodes as 300, 500 and 1000. Each sensor node generates values for nine sensing attributes. Similar to the synthetic data set of 100 sensor nodes, the *tcd* parameter is used to set the temporal correlation of sensor data.

Although the proposed approach is not dependent on a specific routing protocol, in order to study the behavior of the proposed approach according to the routing protocol, the simple *First-Heard-From* (FHF) routing protocol [3] and the GESC routing protocol [20] are used to construct the routing topology. The FHF routing protocol creates the routing tree in such a way that the distance between any two sensor nodes is minimized. Clustering is an effective routing topology construction approach in sensor networks that can increase network scalability and lifetime. Since the GESC routing protocol is an energy-efficient protocol compared with the other clustering protocols for sensor networks, the GESC routing protocol is used to construct the routing topology. The GESC routing protocol exploits the localized network structure and the remaining energy of neighboring sensor nodes to create a routing topology. Since the proposed approach is not dependent on a specific routing protocol, any existing routing protocol can be used. Table 1 summarizes the characteristics of the routing topologies generated by the FHF routing protocol and the GESC routing protocol. Since the results are similar for both routing protocols, the results with the GESC routing protocol are presented in Sections 4.2 and 4.3.

Comparison system: The performance of the proposed approach is compared with TAG [3]. This approach proposed an in-network aggregation to reduce the energy consumption of sensor nodes. In order to process the iceberg query by TAG, intermediate sensor nodes in the routing tree classify tuples into different groups according to the `GROUP BY` clause and apply the aggregate function to the groups. To the best of our knowledge, this is the recent existing approach that can process iceberg queries in sensor networks. Therefore, we choose the TAG as the comparison system.

TABLE 1. Characteristics of routing topologies.

Routing protocol	# of sensor nodes	Maximum # hops from a sensor node to the base station	Average sensor node degree
FHF	100	5	3
	300	9	4
	500	11	6
	1000	15	8
GESC	100	4	3
	300	7	4
	500	11	7
	1000	13	9

Queries: The queries used in our experiments are as follows:

```
SELECT      A1, A2, ..., Ak, count(*)
FROM        Sensors
GROUP BY    A1, A2, ..., Ak
HAVING      count(*) ≥ T
SAMPLE PERIOD 30s
FOR 1day
```

The threshold T varies from 2 to 7. As specified in the `SAMPLE PERIOD` and the `FOR` clauses, the queries are continuously executed every 30 s for 1 day.

Performance metrics: The performance is measured in terms of the energy consumption and the accuracy of the query results. Since the primary source of the energy consumption is the communication, the amount of transmitted data is used to represent the energy consumption. The accuracy of the query results is evaluated using the relative error rate. The accuracy is calculated by

$$accuracy = \left(1 - \frac{unreturned + diffCount}{actual}\right) \times 100,$$

where *unreturned* represents the number of unreturned tuples compared with the real result, *diffCount* represents the number of the returned tuples that have different counts compared with the real result and *actual* represents the cardinality of the real result.

4.2. Energy consumption

Energy consumption on LUCE data set: The energy consumption of sensor nodes is affected by the basic size of data transmitted and the number of transmissions. The transmitted data from the sensor nodes to the base station consist of tuples and their aggregate values. To reduce the basic size of a tuple,

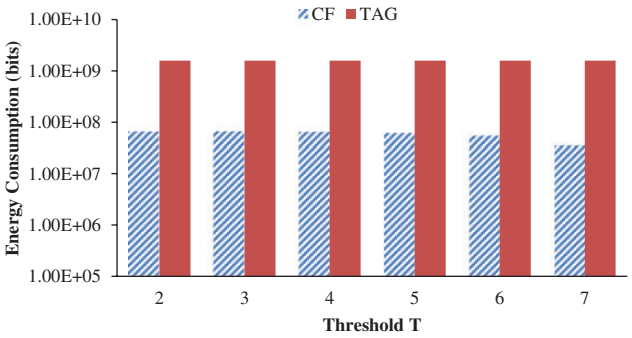

FIGURE 10. Energy consumption comparison between CF and TAG for the LUCE data set.

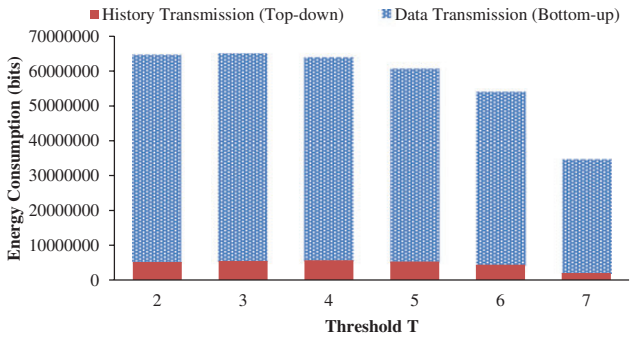

FIGURE 11. Analysis of the energy consumption in CF for the LUCE data set.

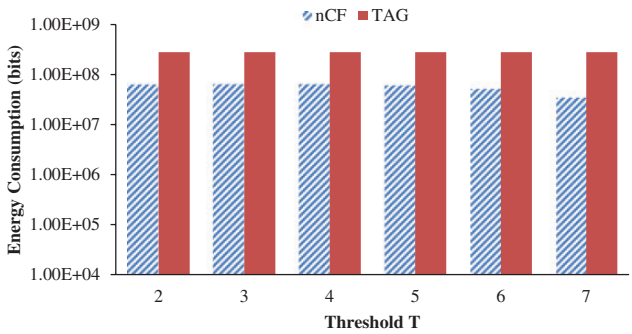

FIGURE 12. Energy consumption comparison between nCF and TAG for the LUCE data set.

our approach proposes the lossless data compression using prime numbers. However, TAG transmits the original tuples themselves. Therefore, the fundamental size of one tuple is different between TAG and our approach, which is denoted as CF. The basic tuple size for TAG and CF is $k \times 64$ bits and 64 bits, respectively (k = # of sensing attributes specified in the query).

Figure 10 shows the energy consumption for various queries over the LUCE data set. Note that the logarithmic scale (base 10) is used for the energy consumption due to the significant performance gap between CF and TAG. If the threshold T has a large value, the number of tuples whose aggregate values are greater than T will be small. Therefore, we can adjust the size of the query result by changing the threshold T. To show the effect of the output size, the threshold value varies from 2 to 7. These parameter values are determined by considering the minimum number of distinct tuples in the LUCE data set.

As shown in Fig. 10, CF consumes less energy than TAG in all cases. TAG can save energy by applying the in-network aggregation. By grouping the distinct tuples and performing the aggregate function (COUNT) on the groups, it can reduce the amount of data transmitted. However, the final aggregate values of the groups cannot be known within the network because sensor nodes are distributed. Therefore, all tuples have to be transmitted to the base station for query processing. However, CF can effectively filter out tuples that will not be included in the query result. By capturing the degree of the temporal correlation of sensor data, CF makes a set of candidate tuples for the future query result and transmits it to sensor nodes. Based on them, sensor nodes can suppress the transmissions of tuples that are not the query answers. Furthermore, CF applies the lossless sensor data compression to reduce the size of one tuple. Therefore, CF can greatly reduce the energy consumption compared with TAG. As the threshold increases, the energy consumption of CF is reduced because the output size becomes smaller than that of the small threshold value. CF effectively filters out unnecessary transmissions regardless of the threshold. However, the energy consumption of TAG is not changed according to the output size because it cannot filter out unnecessary transmissions.

In TAG, data flow in only one direction, from the sensor nodes to the base station. Unlike TAG, in CF, transmissions occur in both directions. To transmit sensor data from the sensor nodes to the base station, bottom-up transmissions occur along the routing tree. To disseminate the estimated future query result to sensor nodes, top-down transmissions occur from the base station to the sensor nodes. The top-down transmissions of the estimated future query result do not occur in TAG. Figure 11 shows an analysis of how much energy is consumed by each transmission in CF. As shown in Fig. 11, the energy consumption by top-down transmissions has a small portion of the overall energy consumption. This confirms that the overhead for transmitting the estimated future query result is low.

Figure 12 shows how much energy is consumed when the advantage of the smaller tuple size is eliminated in CF. In this graph, the logarithmic scale (base 10) is also used for the energy consumption. We use one sensing attribute to make the size of one tuple for TAG and CF the same. Lossless data compression is not used in CF, which is denoted as nCF. This result shows the effect of reducing the energy consumption by our filtering-based data collection itself. Even though the energy consumption of TAG slightly decreases by taking advantage of the reduced one tuple size, nCF continues to consume less energy than TAG.

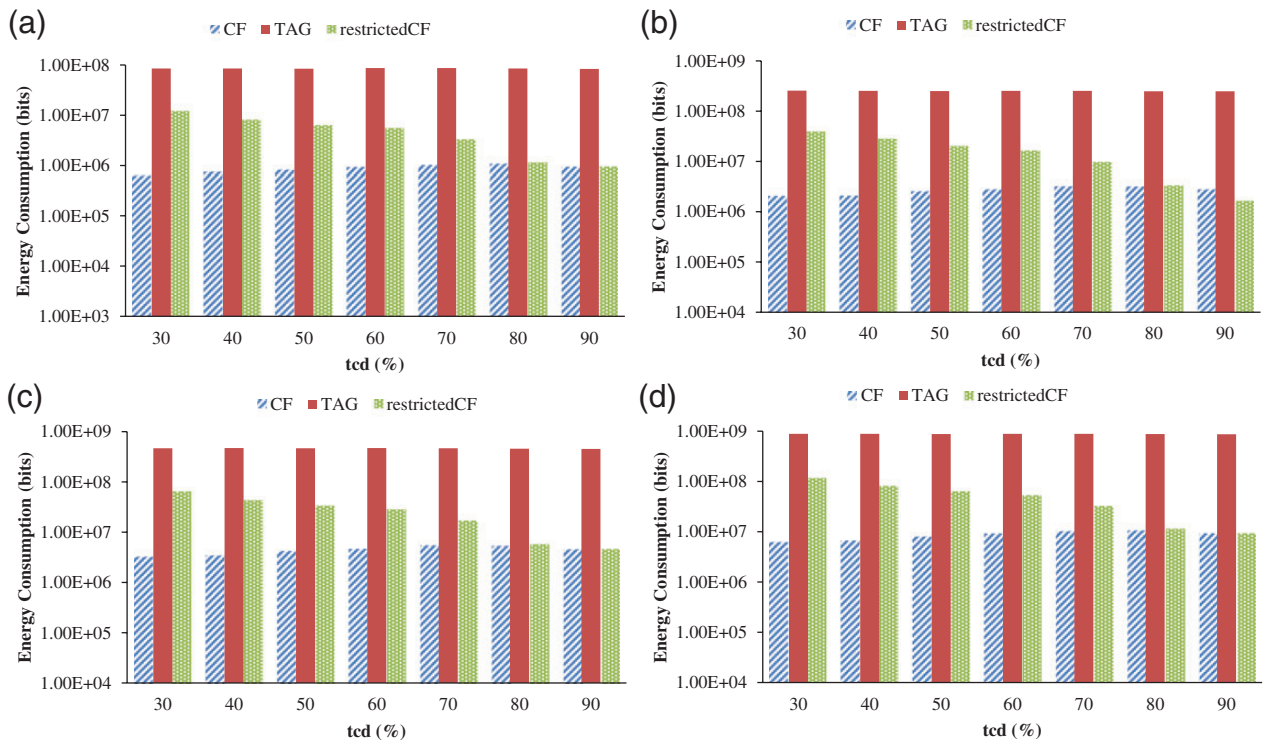

FIGURE 13. Analysis of the energy consumption for synthetic data sets. (**a**) A sensor network of 100 sensor nodes, (**b**) a sensor network of 300 sensor nodes, (**c**) a sensor network of 500 sensor nodes and (**d**) a sensor network of 1000 sensor nodes.

Therefore, our filtering-based data collection is a very energy-efficient method in itself.

Energy consumption on synthetic data sets: In this experiment, we show the energy consumption of CF under different degrees of temporal correlation of sensor data. The number of sensor nodes of the sensor network is varied from 100 to 1000. The degree of temporal correlation of sensor data is controlled by setting the tcd value when making a synthetic data set. A high tcd value represents that sensor data have a strong temporal correlation.

Figure 13 shows the energy consumption for CF, TAG and restrictedCF for sensor networks of 100, 300, 500 and 1000 sensor nodes with various tcd values. The tcd value is varied from 30 to 90%. The threshold value T is set to 5. The logarithmic scale (base 10) is used for the energy consumption in these figures. As shown in Fig. 13, CF consumes less energy than TAG in all cases. The energy consumption of TAG is not affected by the degree of temporal correlation of sensor data. However, in CF, the energy consumption is affected by the tcd values. Since the prediction of the future query result is based on the similarity between two consecutive previous query results, the prediction result is more accurate when sensor data have a strong temporal correlation. Therefore, unnecessary sensor data can effectively be filtered out, which leads to energy saving.

The energy consumption of CF decreases as the tcd value decreases because there are some missing query results by the prediction. To deal with this problem, when the similarity of two consecutive previous query results is below a certain threshold st, the topology information is only used to filter out unnecessary transmissions. When the similarity increases above the st, filtering based on the prediction results is applied. This approach is denoted as restrictedCF in Fig. 13. The st value is empirically set to 0.5 where values for st range from 0 to 1. The energy consumption of restrictedCF increases compared with that of CF because there are cases where the prediction-based filtering cannot be used. In CF, early filtering of unnecessary transmissions is possible based on the predicted query result and hence the energy consumption can be reduced. However, in restrictedCF, for cases when the similarity of two consecutive previous query results is below the st, sensor data have to be transmitted to ancestor sensor nodes until they reach a sensor node that has enough topology information for filtering. Since sensor nodes that are located close to the base station in the routing tree have enough information for filtering based on the network topology, filtering of unnecessary transmissions occurs at these sensor nodes. Thus, in the case of filtering based on the topology information, levels of sensor nodes where filtering is performed are higher than those in CF. Although more energy is consumed compared with CF, restrictedCF can lower the number of missing query results and hence generate more accurate query results than CF. The amount of energy consumption of restrictedCF increases as the tcd value

decreases since there are many cases where the filtering is performed based only on the topology information.

4.3. Accuracy

Accuracy on the LUCE data set: The results of the accuracy on the LUCE data set are given in Fig. 14. Since TAG transmits all tuples of sensor nodes after applying the in-network aggregation, all queries are answered correctly. Therefore, the accuracy of TAG is always 100%. Since sensor nodes are distributed, tuples generated from all sensor nodes have to be transmitted to the base station to find out their global aggregate values. However, a lot of them do not satisfy the `HAVING` condition. Therefore, transmissions of these tuples cause unnecessary waste of energy. To reduce the energy consumption, CF applies the data compression using prime numbers and filtering-based data collection. Since the proposed data compression method can reconstruct the exact original sensor data, this does not affect the accuracy of the query results. Therefore, only the filtering-based data collection affects the accuracy of the query results in CF. To filter out unnecessary transmissions, CF predicts the future query result by modeling the degree of the temporal correlation of sensor data, and uses it to suppress unnecessary transmissions. As shown in Fig. 14, CF provides nearly 100% accurate query results. It means that our prediction model is accurate. Therefore, CF can provide accurate query results while conserving energy.

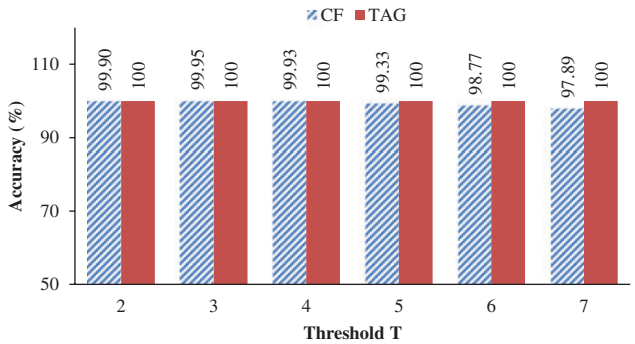

FIGURE 14. Accuracy for the LUCE data set.

Accuracy on synthetic data sets: In this experiment, the accuracies of CF, TAG and restrictedCF for sensor networks of 100, 300, 500 and 1000 sensor nodes are compared under various degrees of the temporal correlation of sensor data. For restrictedCF, the *st* value is set to 0.5. Figure 15 shows the accuracy for various tcd values. The threshold value *T* is set to 5. As shown in Fig. 15, the accuracy of the query results is not affected by the size of the sensor network. The accuracy of the query results in CF decreases as the tcd decreases. For higher tcd values, since sensor data have a strong temporal correlation, the future query result can be accurately predicted by our prediction model. However, for

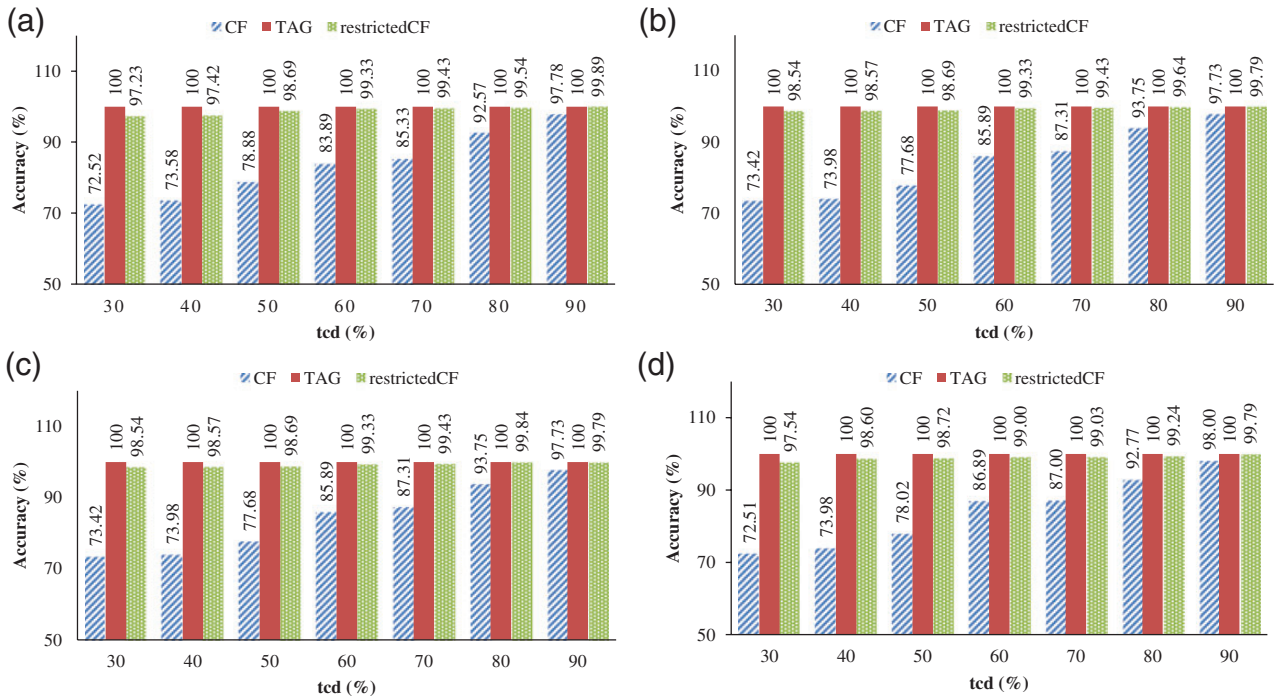

FIGURE 15. Accuracy for synthetic data sets. (**a**) A sensor network of 100 sensor nodes, (**b**) a sensor network of 300 sensor nodes, (**c**) a sensor network of 500 sensor nodes and (**d**) a sensor network of 1000 sensor nodes.

smaller tcd values, due to the randomness of sensor data, there exist some missing sensor data in the predicted future query result, and hence these sensor data are filtered out. restrictedCF can solve this problem by adaptively applying the prediction-based filtering according to the similarity of two consecutive previous query results. Although the energy consumption of restrictedCF increases compared with that of CF, more accurate query results are generated. Therefore, the iceberg query is energy-efficiently and accurately processed by applying the adaptive filtering based on the degree of temporal correlation of sensor data.

5. CONCLUSIONS

To process the iceberg query in a sensor network environment, sensor nodes should transmit their sensor data to the base station because sensor nodes are distributed. The key characteristic of the iceberg query is that the number of the result data is very small relative to a large amount of input data. Therefore, there exist a lot of unnecessary transmissions of tuples that are not to be included in the query result. Since sensor nodes have limited energy resources, it is impractical to transmit all sensor data to the base station for query processing.

This paper proposes an energy-efficient iceberg query processing technique in sensor networks. Since the energy consumption of sensor nodes depends on the size of data transmitted and the number of transmissions, we devise the filtering-based data collection method based on lossless data compression. Usually, the number of sensing attributes specified in the query is large and so is the size of one tuple. Based on the Fundamental Theorem of Arithmetic in the number theory, we propose the data compression method that reduces the size of data transmitted to the size of one sensing attribute regardless of the number of sensing attributes specified in the `SELECT` clause. Therefore, we can reduce the energy consumption caused by the size of data transmitted. The number of transmissions is greatly reduced by our filtering-based data collection method. Since sensor data generally exhibit strong temporal correlations, a large portion of the previous query result will be generated in the following sample period. The degree of the temporal correlation between two consecutive previous query results is represented well by our proposed model. Based on this model, the possible future query result is accurately predicted. The sensor node can filter out unnecessary transmissions by comparing the intermediate result with the predicted future query result. Therefore, we can greatly reduce the energy consumption caused by unnecessary transmissions.

Experiments are conducted to evaluate the performance of the proposed approach by using real-world and synthetic data sets. The results show that the proposed approach outperforms the comparison system in terms of energy consumption. In addition, the proposed approach provides nearly 100% accurate query results.

In general, sensor data also have strong spatial correlations. Sensor data of geographically proximate sensors are likely to be highly correlated. Therefore, as future work, we will conduct a study to improve our approach by exploiting the spatial correlation of sensor data.

FUNDING

This work was supported in part by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD110006MD, Korea and in part by the National Research Foundation of Korea grant funded by the Korean government (MSIP) (No. NRF-2009-0081365).

REFERENCES

[1] Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R. and Ullman, J.D. (1998) Computing Iceberg Queries Efficiently. *Proc. VLDB '98*, San Francisco, CA, USA, pp. 299–310. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[2] Chu, D., Deshpande, A., Hellerstein, J.M. and Hong, W. (2006) Approximate Data Collection in Sensor Networks Using Probabilistic Models. *Proc. ICDE '06*, Atlanta, GA, USA, pp. 48–59. IEEE Computer Society, Washington, DC, USA.

[3] Madden, S., Franklin, M.J., Hellerstein, J.M. and Hong, W. (2002) Tag: a tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Oper. Syst. Rev.*, **36**, 131–146.

[4] Madden, S.R., Franklin, M.J., Hellerstein, J.M. and Hong, W. (2005) Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, **30**, 122–173.

[5] Silberstein, A., Braynard, R. and Yang, J. (2006) Constraint Chaining: On Energy-Efficient Continuous Monitoring in Sensor Networks. *Proc. SIGMOD '06*, Chicago, IL, USA, pp. 157–168. ACM, New York, NY, USA.

[6] Manjhi, A., Shkapenyuk, V., Dhamdhere, K. and Olston, C. (2005) Finding (Recently) Frequent Items in Distributed Data Streams. *Proc. ICDE '05*, Tokyo, Japan, pp. 767–778. IEEE Computer Society, Washington, DC, USA.

[7] Zhao, Q.G., Ogihara, M., Wang, H. and Xu, J.J. (2006) Finding Global Icebergs Over Distributed Data Sets. *Proc. PODS '06*, Chicago, IL, USA, pp. 298–307. ACM, New York, NY, USA.

[8] Zhao, H.C., Lall, A., Ogihara, M. and Xu, J. (2010) Global Iceberg Detection Over Distributed Data Streams. *Proc. ICDE '10*, Long Beach, CA, USA, pp. 557–568. IEEE Computer Society, Washington, DC, USA.

[9] Sharaf, A., Beaver, J., Labrinidis, A. and Chrysanthis, K. (2004) Balancing energy efficiency and quality of aggregate data in sensor networks. *VLDB J.*, **13**, 384–403.

[10] Yao, Y. and Gehrke, J. (2003) Query Processing in Sensor Networks. *Proc. CIDR '03, CIDR Conf.*, Asilomar, CA, USA, pp. 233–244. Asilomar, CA, USA.

[11] Deshpande, A., Guestrin, C., Madden, S.R., Hellerstein, J.M. and Hong, W. (2004) Model-Driven Data Acquisition in Sensor Networks. *Proc. VLDB '04*, pp. 588–599. VLDB Endowment Inc., San Jose, CA, USA.

[12] Shrivastava, N., Buragohain, C., Agrawal, D. and Suri, S. (2004) Medians and Beyond: New Aggregation Techniques for Sensor Networks. *Proc. SenSys '04*, Baltimore, MD, USA, pp. 239–249. ACM, New York, NY, USA.

[13] Sayood, K. (2000) *Introduction to Data Compression* (2nd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[14] Guestrin, C., Bodik, P., Thibaux, R., Paskin, M. and Madden, S. (2004) Distributed Regression: An Efficient Framework for Modeling Sensor Network Data. *Proc. IPSN '04*, Berkeley, CA, USA, pp. 1–10. ACM, New York, NY, USA.

[15] Deligiannakis, A., Kotidis, Y. and Roussopoulos, N. (2004) Compressing historical information in sensor networks. *Proc. SIGMOD '04*, Paris, France, pp. 527–538. ACM, New York, NY, USA.

[16] Heinzelman, W.R., Kulik, J. and Balakrishnan, H. (1999) Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. *Proc. MOBICOM '99*, Seattle, WA, USA, pp. 174–185. ACM, New York, NY, USA.

[17] Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J.S. and Silva, F. (2003) Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, **11**, 2–16.

[18] Heinzelman, W.B., Chandrakasan, A.P. and Balakrishnan, H. (2002) An application-specific protocol architecture for wireless microsensor networks. *IEEE Trans. Wirel. Commun.*, **1**, 660–670.

[19] Younis, O. and Fahmy, S. (2004) Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Trans. Mob. Comput.*, **3**, 366–379.

[20] Dimokas, N., Katsaros, D. and Manolopoulos, Y. (2010) Energy-efficient distributed clustering in wireless sensor networks. *J. Parallel Distrib. Comput.*, **70**, 371–383.

[21] Silberstein, A.S., Braynard, R., Ellis, C., Munagala, K. and Yang, J. (2006) A Sampling-Based Approach to Optimizing Top-k Queries in Sensor Networks. *Proc. ICDE '06*, Atlanta, GA, USA, pp. 68–78. IEEE Computer Society, Washington, DC, USA.

[22] Sadler, C.M. and Martonosi, M. (2006) Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks. *Proc. SenSys '06*, pp. 265–278. ACM, New York, NY, USA.

[23] Welch, T.A. (1984) A technique for high-performance data compression. *Computer*, **17**, 8–19.

[24] Marcelloni, F. and Vecchio, M. (2009) An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks. *Comput. J.*, **52**, 969–987.

[25] Wu, X., Lee, M.L. and Hsu, W. (2004) A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. *Proc. ICDE '04*, Boston, MA, USA, pp. 66–78. IEEE Computer Society, Washington, DC, USA.

[26] Lee, C.-H. and Chung, C.-W. (2011) RFID data processing in supply chain management using a path encoding scheme. *IEEE Trans. Knowl. Data Eng.*, **23**, 742–758.