An Energy-Efficient Technique for Processing Sensor Data in Wireless Sensor Networks

Kyung-Chang Kim[1,1] and Choung-Seok Kim[2]

[1] Dept. of Computer Engineering, Hongik University
Seoul, Korea
[2] Dept. of Information Technology, Silla University
Pusan, Korea
[1]kckim@hongik.ac.kr, [2]cskim@silla.ac.kr

Abstract. Many wireless sensor network (WSN) applications require a join of sensor data between sensor nodes. In this paper, we introduce an energy-efficient join technique for sensor networks that minimizes the communication cost. In WSN, several nodes need to collaborate to carry out the join since the processing capabilities of a single node is limited. Our novel algorithm is based on the join of sensor data stored in column-oriented databases. A column-oriented database store data in column order (i.e. column-wise) and not in row order as in traditional relational databases. The proposed algorithm is energy-efficient since only the minimum necessary data are shipped to the join region for final join processing. The performance analysis show that the proposed algorithm performs better than the traditional join algorithms, such as the SNJ and RFB algorithms, that are based on relational databases.

Keywords: Wireless sensor network, join technique, column-oriented database, communication cost, performance analysis.

1 Introduction

The advancement of hardware technology and the desire for new application areas have enabled the widespread use and deployment of sensor networks. A **wireless sensor network (WSN)** consists of spatially distributed autonomous sensor nodes to *monitor* physical or environmental conditions, such as temperature, pressure, vibration, sound, humidity, motion or pollutants and to cooperatively pass their data through the network to a main location. Each sensor node is capable of observing the environment using sensors, storing the observed values, processing them and exchanging them with other nodes over the wireless network [1]. The topology of the WSNs can vary from a simple star network to an advanced multi-hop wireless mesh network.

[1] This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number 2011-0025995).

Many sensor network applications require a join of sensor data scattered among sensor nodes. For example, in a vehicle tracking system, we may be interested in vehicles that entered and exited a designated length of a road to monitor the traffic volume and speed. In the conventional in-network approach, several sensor nodes collaborate to perform the join. The result of the join is then transmitted to the base station.

In this paper, we propose a novel join technique for wireless sensor networks where the sensor data are stored in column-oriented databases. Recent years have seen an increased attention and research work on column-oriented databases. A column-oriented database store data in column order (i.e. column-wise) and not in row order as in traditional relational databases. They are more I/O efficient for read-only queries since they only access those columns (or attributes) required by the query. The read-only queries are common in workloads and applications found in WSN.

From an energy-efficient perspective, a join technique must minimize the communication cost of transmitting sensor data from designated sensor nodes to the join region within the network to perform the join. Minimizing the communication cost is important since the biggest use of battery power, the only energy source. Through performance experiments, we show that the communication cost incurred in our proposed technique is lower than similar techniques mentioned in the literature.

The rest of the paper is organized as follows. We discuss related works in Section 2. In Section 3, we present a novel join algorithm based on early materialized strategy in column-oriented databases. The query performance of the proposed algorithm is compared with traditional join algorithms for sensor network databases in Section 4. The conclusion is given in Section 5.

2 Related Work

Several technique to handle simple joins in sensor networks are proposed in the literature [3,4,6]. The general join strategies in sensor networks can be classified as naïve join, sequential join, and centroid join depending on the location of the join region in the sensor network. These join strategies all perform in-network join processing, where the actual join is performed in the network by several collaborating sensor nodes called the join region. The main problem with these general approaches is the communication cost overhead associated with low join selectivity. Tuples that are not candidates for join can be transmitted to the join region.

A better approach is to transmit only those readings that are likely to contribute to the join results. Several filtering techniques were proposed where only those sensor data involved in the join result are transmitted to the join region near the base station. One approach is the synopsis join (SNJ) algorithm [7]. The key idea is to use synopsis of sensor readings to prune those readings that are irrelevant to join results. Another technique is the record filtering using bit-vector (RFB) algorithm [5]. The RFB algorithm uses bit vectors produced after semi-join is performed to prune unnecessary data before shipping data from each node to the join region. The RFB algorithm is all based on traditional row-oriented databases where sensor readings are taken and stored in relational databases.

3 An In-Network Join Technique for WAN

In this paper, we focus on the processing of BEJ (Binary Equi-Join) queries. A BEJ query only involves an equi-join (join involving equality join condition) of two relations R and S whose data are scattered among sensor nodes in the R region (for table R) and the S region (for table S). A BEJ query is initiated at the sensor node called query sink and the query result is also collected at query sink. Since the memory size at each node is limited, the sink node is unable to perform the join locally. The join has to be performed through the collaboration of several nodes called the ***join region***.

We propose an in-network join technique based on the early materialized strategy for column-oriented databases [2]. In the early materialization strategy, when a column is accessed, the column is added to the intermediate query result (to form tuples) if the column is needed. In a late materialization strategy, the accessed columns do not form tuples until after some part of the query plan has been processed.

The proposed join algorithm executes in three phases, namely the ***selection*** phase, the ***join*** phase, and the ***result*** phase. In the selection phase, the semi-join of join columns of R and S is performed to determine the qualified columns and column values for the given query. The columns in R and S participating in the query result are stitched to form tuples before shipping to the join region for final join. In the join phase, the stitched tuples of R and S are joined using either a nested-loop or a sort-merge join algorithm. In the result phase, the result of join performed in the join region is sent to the query sink as the query result.

We describe the three phases in more detail using an example schema and query. The example schema is R(Sid, Vid, Time, Speed) and S(Sid, Vid, Time, Speed) and the example query is *Select R.Vid, R.Time, S.Time From R, S Where R.Vid=S.Vid*. The example query monitors the time interval for vehicles moving from region R to region S to determine the time spent to move from region R to region S.

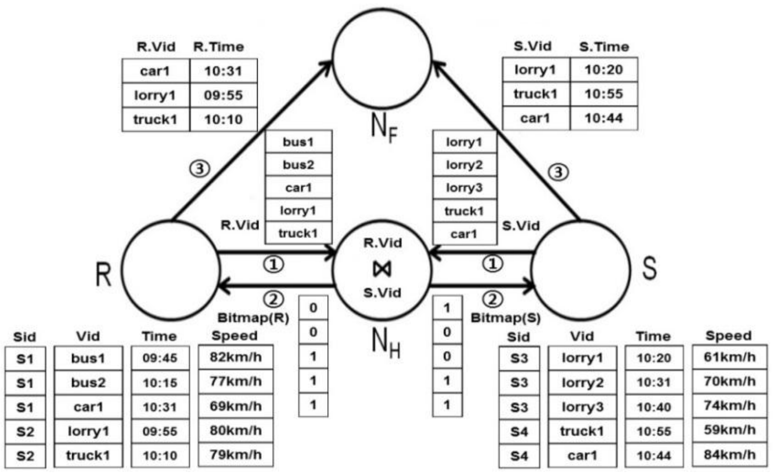
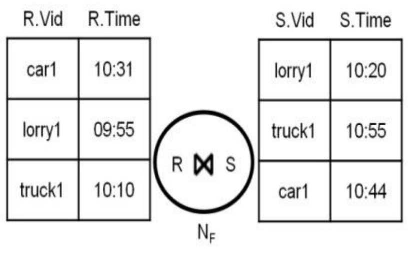


Figure 1: The Selection Phase

The execution of the selection phase is shown in Fig. 1 which also shows the example relation instances for both R and S. It can be seen that the columns of R and S are stored separately in a column-oriented database. In step 1, the join column of R, which is R.Vid column, is shipped to the semi-join region N_H. Likewise, the join column of S, which is S.Vid, is also shipped to N_H. The semi-join region N_H is chosen to reduce the cost of joining R.Vid and S.Vid columns. A join (i.e semi-join) is performed between the R.Vid and S.Vid columns. In step 2, the result of the semi-join is used to create bitmaps to be shipped back to region R and region S. A Bitmap(R) contains one bit for every tuple in R.Vid. That bit is set to 1 if it is in the semi-join result performed in N_H. Likewise a Bitmap(S) contains one bit for every tuple in S.Vid. That bit is set to 1 if it is in the join result performed in N_H. Bitmap(R) and Bitmap(S) are shipped back to region R and region S respectively.

In regions R and S, the column values that are involved in the query are stitched (i.e. tuple reconstruction) before shipping to the join region. In region R, the corresponding column values of R.Vid and R.Time are stitched to form tuples. Only the column values whose corresponding bits are set to 1 in Bitmap(R) are selected for stitching. Likewise, in region S, the corresponding column values of S.Vid and S.Time are stitched to form tuples. Only the column values whose corresponding bits are set to 1 in Bitmap(S) are selected for stitching. In step 3, all stitched tuples of R and S are shipped to the join region N_F.

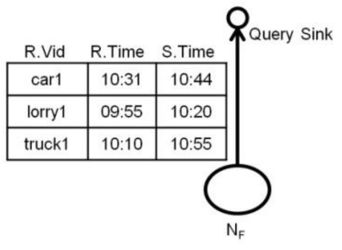
In the join phase, the stitched tuples of R and S are joined in join region N_F to get the join result. The join region N_F is chosen to reduce the cost of transmitting data from region R and region S to the query sink. Either a nested-loop or a sort-merge join algorithm can be used to perform the final join. The execution of the join phase is shown in Fig. 2. In the result phase, the join result produced in join region N_F is shipped to the query sink, which gathers and sends the query result back to the user. The execution of the result phase is shown in Fig. 3.



R.Vid	R.Time
car1	10:31
lorry1	09:55
truck1	10:10

S.Vid	S.Time
lorry1	10:20
truck1	10:55
car1	10:44

Figure 2: The Join Phase



R.Vid	R.Time	S.Time
car1	10:31	10:44
lorry1	09:55	10:20
truck1	10:10	10:55

Figure 3: The Result Phase

The proposed algorithm is energy-efficient since only the columns and the column values necessary for processing the join query are shipped to the sensor nodes in the sensor network. Unlike the late materialized strategy, the proposed strategy does not require shipping position lists, containing the positions of the column values to be joined, to the join region to perform the join. Compared to traditional algorithms based on relational databases, the cost of shipping data to the N_H and N_F region is minimized.

4 Performance Analysis

To test the cost-effectiveness of the join algorithm proposed in this paper, a performance analysis is conducted and the performance comparison is made with the existing SNJ and RFB algorithms.

4.1 Experiment Environment

In reality, both table R and table S are distributed among several nodes in the WAN. In addition, the semi-join region N_H and the final join region N_F consist of several collaborating nodes. Without loss of generality, for simplicity, we compute the communication cost assuming that R, S, N_H, N_F are all stored in a single node respectively.

In order to simplify network traffic analysis, we assumed that no failure occurs during message transmission. The size of the message and the tuple was assumed to be 40 bytes each. For the given BEJ query, the size of the resulting join tuple (i.e. query result) was assumed to be 30 bytes. For column-oriented database, the size of each column was assumed to be 10 bytes.

4.2 Experiment Result

The left graph of Fig. 4 shows the communication cost of the join algorithms for different join selectivity and when the cardinality of table R and table S are 2000 tuples each. The join selectivity selected in the experiment was varied to include 0.01, 0.1, 0.3 and 0.5. Join selectivity of 0.01 means that only 1% of tuples satisfies the join condition.

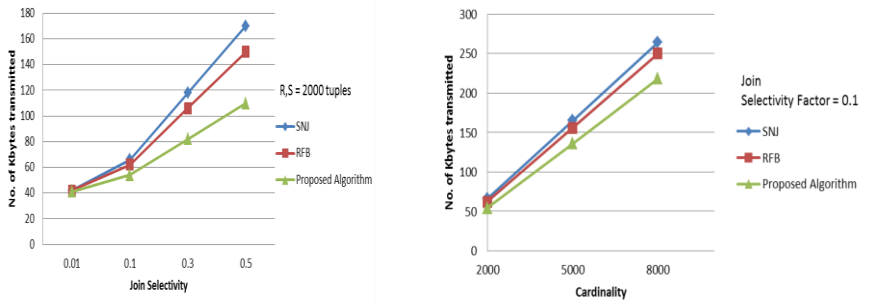


Figure 4: Communication Cost on Join Selectivity, Cardinality

Compared to the SNJ algorithm, the communication cost decreases from 2.5% to 35% for join selectivity varied from 0.01 to 0.5. Compared with RFB algorithm, the communication cost decreases from 2.5% to 27% for join selectivity varied from 0.01 to 0.5. As can be seen, the performance of the proposed algorithm gets better as the join selectivity gets lower.

The right graph of Fig. 4 shows the communication costs for different cardinalities of the tables to be joined. The join selectivity used is 0.1 and the cardinalities are varied from 2000 tuples each for R and S to 5000 tuples each and 8000 tuple each. Our proposed algorithm outperforms both the SNJ and the RFB algorithms for all different cardinalities tested. Compared to the SNJ algorithm, the communication cost decreases around 18% for all cardinalities. Compared to the RFB algorithm, the communication cost decreases around 13% for all cardinalities. The performance of our proposed algorithm is better but the communication cost remains the same for different cardinalities. The performance result shows that the join selectivity determines the communication cost more than the size of the tables to be joined.

5 Conclusion

In this paper, we proposed an algorithm for processing data in sensor networks. The proposed algorithm is based on an early materialized strategy in column-oriented database where the data tables are stored in columns. The proposed algorithm is energy-efficient since only those columns and column values involved in a query are shipped between the sensor nodes.

Experimental results show that our proposed join algorithm outperforms both the SNJ and the RFB algorithms in reducing the communication cost. Both the SNJ and the RFB algorithms are based on relational databases. To validate our experiment, we used different values of join selectivity as well as different table sizes. In performance analysis, we showed that the performance of the proposed algorithm gets better as the join selectivity decreases. If more tuples are joined and output in the join result, the communication cost of the proposed algorithm decreases. However, for a given join selectivity, the table size does not affect the performance.

References

1. Abadi, D.J., Madden, S., Lindner, W.: REED: robust, efficient filtering and event detection in sensor networks. In: Proceedings of the VLDB (2005)
2. Abadi, D.J., Myers, D.S., Dewitt, D. J. and Madden, S.R.: Materialization strategies in a column-oriented DBMS. In: Proc. of International Conference on Data Engineering, pp.466-475, 2007
3. Chowdhary, V., Gupta, H.: Communication-Efficient Implementation of Join in Sensor Networks. In: Proceedings of DASFAA, Beijing, China (2005)
4. Coman, A., Nascimento, M., Sander, J.: On join location in sensor networks. In: Proceedings of MDM (2007)
5. Kim, K.C and Oh B. J.: An Energy-Efficient Filtering Approach to In-Network Join Processing in Sensor Network Databases. In: Proceedings of Multimedia, Computer Graphics and Broadcasting, Jeju, Korea, 2011
6. Madden, S.: The Design and Evaluation of a Query Processing Architecture for Sensor Networks. Ph.D. Thesis, UC Berkeley (2003)
7. Yu, H., Lim, E., Zhang, J.: On In-network Synopsis Join Processing for Sensor Networks. In: Proceedings of MDM (2006), Nara, Japan