Research Article

An Efficient Processing of Join Queries for Sensor Networks Using Column-Oriented Databases

Kyung-Chang Kim

Department of Computer Engineering, Hongik University, Seoul 121-791, Republic of Korea

Correspondence should be addressed to Kyung-Chang Kim; kckim@hongik.ac.kr

Received 10 May 2013; Revised 10 September 2013; Accepted 14 September 2013

1. Introduction

Recent developments in hardware technology have enabled the widespread use and deployment of sensor networks. A sensor network consists of a large number of sensor nodes that combine physical sensing capabilities such as temperature, noise, light, or seismic sensors with networking and computation capabilities. Each sensor node is capable of observing the environment using sensors and storing the observed values, processing them and exchanging them with other nodes over the wireless network. There are many applications of sensor networks including vehicle tracking, environmental monitoring, and warehouse management.

Most of the sensor nodes are battery operated, which has limited power supply and cannot be replaced. The energy cost for communication with neighboring nodes is much higher than the energy required for sensing or computation within the node. Hence, to reduce battery consumption, it is necessary to reduce communication with neighboring nodes. In traditional databases, it is important to minimize the I/O cost during query processing. However, in sensor network, minimizing the communication cost is the main performance criteria. In this paper, the focus is on proposing an energy efficient processing of join queries over sensor networks.

Users query the sensor network to manipulate the sensed data on the monitored environment. A sensor network application, such as environmental monitoring, requires joining sensor data scattered over the network. A naïve way to answer an ad hoc join query for such applications is to move the sensor readings back to the base station and perform the join at the base station. This approach may incur high communication cost since all sensor readings have to be transmitted to the base station resulting in high consumption of limited battery power at each node.

A better approach is to transmit only those readings (data) that are likely to contribute to the join results. Less data shipped means lower communication cost, resulting in less battery power consumption and saving battery energy. One way is to use a filtering technique like a semijoin or a bit vector to transmit only relevant sensor data involved in the join result to the join region near the base station. The key idea is to use synopsis of sensor readings to prune those readings that are irrelevant to join results. The other approach is to use bit vectors instead of synopsis to prune irrelevant
data. These approaches typically deal with data that are stored in traditional relational databases. Storing data in relational databases means that all the tuples or records that are directly involved in a join must be sent to the final join region. This results in shipping unnecessary column values in a tuple.

To solve this problem, we propose an energy-efficient join method based on column-oriented databases. The proposed join method deals with data that are stored in column-oriented databases. Recent years have witnessed a significant amount of attention and research work on column-oriented database systems. Column-oriented databases store data columnwise (in columns) unlike traditional relational databases that store data rowwise (in rows). Column-oriented databases are more I/O efficient for read-only queries since they read from disk only those attributes (or columns) accessed by the query. The read-only queries are common in workloads found in sensor networks and data warehouse systems.

Figure 1 shows the storage structure of a column-oriented and a row-oriented database. Most traditional DBMS, including Oracle, are based on row-oriented databases. A major drawback of a row-oriented database for read-only queries is that columns not related to the query result must also be read in from disk. However, column-oriented databases are not suitable for update queries since several I/O operations are required to store a single tuple. This is because a database tuple with several columns has to be stored separately column by column in a column-oriented database.

The main contribution of this paper is that we developed an energy-efficient technique for processing join queries over sensor networks using column-oriented technology. We believe that our work is the first in using column-oriented database technology over sensor networks during query processing. The rest of the paper is organized as follows. We discuss related joint processing techniques for sensor network databases in Section 2. In Section 3, we present a join algorithm for sensor network based on a column-oriented database. Performance analysis is conducted in Section 4 to compare the query performance with existing join algorithms. The conclusion is given in Section 5.

2. Related Works

Recently, query processing methods have been proposed for wireless sensor networks. Many deal with techniques for efficient processing of aggregate queries [1–3]. Other proposals deal with energy-efficient methods to reduce energy consumption during query processing [4–6]. The general strategies for join query processing in sensor networks can be classified depending on the location of the join region in the sensor network [7]. 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.

There are several join techniques for sensor networks where tuples not participating in the join result are pruned and not transmitted to the final join region. One such approach is the synopsis join (SNJ) algorithm [8]. In this strategy, which is similar to a semijoin, data not contributing to the join results are pruned in the early stage of the join processing. Another approach is the record filtering using bit vector (RFB) algorithm [9]. The RFB algorithm is a hybrid algorithm that reduces the size of data to be transmitted to the join region using bit vectors. An incremental join algorithm (IJA) for sensor networks is proposed in [10] to reduce the overhead of moving data between sensor nodes. The join algorithms discussed so far all store the sensor data (readings) in the sensor nodes in traditional row-oriented databases.

Prior research has suggested that important optimization techniques exist for column-oriented databases. One such technique is late materialization, where columns that are read off disk are joined together into tuples (i.e., tuple reconstruction) as late as possible in a query plan [11]. Another technique is PAX [12], which clusters values of a column together, in a database page with row-store records, as column-based layout. The invisible join [13] extends previous work on improving performance of star schema joins by taking advantage of the column-oriented layout. It is an efficient late materialized join, but minimizes the values that need to be extracted out of order. However, no such optimization techniques exist for sensor network applications using column-oriented databases.

3. Join Algorithm

The data collected in the sensor network can be seen as one relation distributed over the sensor nodes, called the sensor relation or table. Traditional optimization techniques for query processing in sensor networks use filtering techniques such as semijoin or bit vector to reduce shipping data to the final join region. However, these methods, which are based on relational databases, must ship all the tuples or records that are directly involved in a join to the final join region. This results in shipping unnecessary column values in the shipped tuples. In this paper, we propose an energy-efficient join method based on column-oriented databases that further reduce communication cost by shipping only relevant columns of tuples to be joined.

The proposed technique is the use of the late materialized join technique in column-oriented databases [11]. We assume that the sensor table is stored in a column-oriented database. A simple nondistributed version of this join technique involving two sensor tables to be joined is described as follows.
The join column of the first sensor table is joined with the join column from the second sensor table. The result of this join is a position list containing two sets of positions: one from the first sensor table and the other from the second sensor table. In general, at most one of the positions in the position list is produced in sorted order.

The two sets of positions in the position list can be created using Algorithm 1. In general, the join method used is the nested-loop join (NL join).

Values from the relevant sensor table columns at this set of positions are then extracted. These values are then stitched into a tuple and included to the final join result. The selection and join operations associated with a given query are executed for each column individually. The qualified column values are stitched into a tuple (i.e., tuple reconstruction) just before the query result is shipped to the query sink.

In this paper, we focus on the processing of a binary equi-join (BEJ) query, a two-way join involving just two relations (or tables). A BEJ query is initiated at the sensor node called the query sink, and the query result is also collected at the query sink. The query sink sends the requested query to those sensor nodes that contain sensor data (or readings) involved in the query. 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 among several nodes called the join region. In addition, the data in sensor tables are distributed among sensor nodes in the sensor network and are physically stored using a column-oriented database.

The proposed join algorithm is an in-network hybrid algorithm that reduces the size of data to be transmitted to the join region using a bitmap. In addition, physically storing the sensor data in columns, and not rows, enables shipping only those columns involved in the query. This further reduces the size of data to be shipped to the final join region. An in-network join processing means that join is performed among the sensor nodes contained in the sensor network. Shipping all the sensor data (or readings) to a server outside the sensor network for join incurs excessive communication cost. Hence, in-network join processing is necessary to reduce communication cost. The strategy taken in our algorithm is similar to those of both the SNJ and the RFB algorithms except that we use a column-oriented database to store the sensor data.

The symbols used in describing the proposed algorithm are shown in Table 1. Figure 2 shows the late materialized join strategy of the proposed algorithm to join data in sensor nodes located in region R (i.e., sensor table R) and region S (i.e., sensor table S). The sensor data (or readings) in each sensor node is physically stored in a column-oriented database. This means that the table values are physically stored in column order instead of record or tuple order as in conventional relational databases. Without loss of generality, we assume that there is a routing protocol that maps a node in one region to a node in another region of the sensor network in transmitting data.

A sensor network database is similar to a distributed database, where the data table is distributed among geographically scattered sites. It is a well-known fact that semijoin increases the performance of join processing in distributed databases by reducing the amount of data to be shipped for final join. In the proposed join strategy, after the semijoin is performed, only the qualified tuples to be added to the join result are shipped to the join region for final join.
The nodes in the $N_R$ region, where semijoin is performed, are selected using the network topology and data distribution information for region R and S. The central node of $N_R$ is the node that minimizes the routing communication cost of the semi-tables $P_R$ and $P_S$. In addition, the nodes in the $N_R$ region, where the final join is performed, are selected such that the central node of $N_R$ is the minimum point in the triangle formed between the central nodes of region R, region S, and the query sink.

In the proposed join algorithm, the join of sensor table R (in region R) and sensor table S (in region S) is performed in four steps, namely, the semijoin step, the selection step, the join step, and the result step. In the semijoin step, a semijoin is performed in the semijoin region that produces qualified join tuple positions in R and S. In the selection step, the column values in the qualified columns of R and S are shipped to the join region. The column values selected in R and S are based on the qualified join tuple positions in R and S that are produced in the semijoin step. The qualified columns selected depend on the query to be executed. In the join step, the actual join is performed in the join region. In the result step, tuple stitching or reconstruction is performed to produce tuples in the join result. Tuple stitching is done by referring to the qualified join tuple positions produced in the selection step. The details of the four steps of the proposed join strategy are shown in Figure 2.

3.1. The Semijoin Step. The objective of the semijoin step is to perform semijoin to prune those tuples not involved in the join. This eliminates sending unnecessary tuples from regions R and S to the join region, thereby reducing communication cost during join processing. The execution of the semijoin step is as follows. The semi-table $P_R$ in table R, which is the projection of the join column in R, is shipped to region $N_{H}$. $P_R$ contains the join column values of table R that joins with table S. Likewise, the semi-table $P_S$ in table S, which is the projection of the join column in S, is shipped to region $N_H$. $P_S$ contains the join column values in table S that joins with table R. Region $N_H$ is the semijoin region that performs the semijoin of $P_R$ and $P_S$. Two sets of bitmap are created after the join of $P_R$ and $P_S$ in region $N_H$. A Bitmap($R$) contains one bit for every entry in $P_R$. That bit is set to 1 if it is in $P_R$. Likewise, a Bitmap($S$) contains one bit for every entry in $P_S$. That bit is set to 1 if it is in $P_S$. Bitmap($R$) and Bitmap($S$) contain the qualified join tuple positions in tables R and S, respectively. Bitmap($R$) and Bitmap($S$) are shipped back to regions R and S respectively.

3.2. The Selection Step. Only the qualified columns and qualified column values are shipped to the join region for final join. The qualified columns depend on the query, specifically the projection operator, to be executed and the qualified column values depend on the bitmap produced in the semijoin step. For each qualified column in R, which includes the join column, the qualified column values $P_{R'}$ are shipped to join region $N_{R'}$. Bitmap($R$) is used to extract qualified column values $P_{R'}$ in table R satisfying the join condition. The qualified column values correspond to the positions in bitmap where the bit is set to 1. Likewise, for each qualified column in S, which includes the join column, the qualified column values $P_{S'}$ are shipped to join region $N_{S'}$. Bitmap($S$) is used to extract qualified column values $P_{S'}$ in table S satisfying the join condition. Several $P_{R'}$ and $P_{S'}$ are shipped to the join region depending on the qualified columns. We assume that, for several $P_{R'}$ ($P_{S'}$) shipped to the join region, the positions of column values in each $P_{R'}$ ($P_{S'}$) shipped to the join region have the same tuple position as in original table (R/S).

3.3. The Join Step. In the join step, the qualified column values of the join columns of R and S that are shipped to the join region $N_R$ are used to perform the nested-loop join. The result of the join is two sets of position lists: one for the join column of R and the other for the join column of S. Each entry pair in the position lists contains the positions of join columns of R and S with the same join attribute value.

3.4. The Result Step. In the result step, positions associated with each entry pair in the position lists created in the join step are used to extract values from the selected columns (including the join column) that are shipped from region R and region S in the selection step. The values extracted from the qualified columns of R and S are stitched (i.e., tuple reconstruction) together to form tuples. The stitched tuples, which are the query result, are then shipped to the query sink. The proposed algorithm is a late materialized column-oriented strategy since tuple reconstruction is done after the join.

3.5. A Running Example. The details of the proposed join algorithm are further explained through the execution a BEJ query and the example schema. An example of a BEJ query we use is shown below:

Select R.Vid, R.Time, S.Time
From Region R, S
Where R.Vid = S.Vid

The schema for sensor table R is R (Sid, Vid, Time, and Speed) and the schema for sensor table S is S (Sid, Vid, Time, and Speed). We assumed that both tables R and S are distributed. They are co-relation sensor tables in regions R and S. In a column-oriented database, the tables are physically stored in columns and not rows. This BEJ query for sensor networks monitors a certain vehicle (identified by Vid) that travels from one region (i.e., region R) to another region (i.e., region S) in the sensor network. The Time attribute contains values corresponding to when a particular vehicle was detected in the region, and the Speed attribute contains values corresponding to the speed of a particular vehicle travelling in the region.

Figure 3 shows the semijoin and selection step of the query execution. In the example schema, $P_R$ in region R is the column Vid in R. The values of the Vid column of table R are shipped to region $N_{R'}$. Likewise, $P_S$ in region S is column Vid in S. The values of Vid column of table S are shipped to region...
The semijoin of $P_R$ and $P_S$ creates $\text{Bitmap}(R)$ which is 00111 and $\text{Bitmap}(S)$ which is 10011. $\text{Bitmap}(R)$ is shipped back to region $R$ and is used to create qualified column values $P_{R'}$ from Vid column in region $R$. The qualified values in $P_{R'}$ are the values in positions 3, 4, and 5 according to $\text{Bitmap}(R)$. Likewise, $\text{Bitmap}(S)$ is shipped back to region $S$ and is used to create the qualified column values $P_{S'}$ from Vid column in region $S$. The qualified values in $P_{S'}$ are the values in positions 1, 4, and 5 according to $\text{Bitmap}(S)$.

In the example query, the qualified columns of $R$ and $S$ to be shipped to the join region $N_F$ are columns Vid and Time. The qualified values $P_{R'}$ in the Time attribute (or column) of $R$ to be shipped to the join region $N_F$ are the values in positions 3, 4, and 5 according to $\text{Bitmap}(R)$. The qualified values $P_{S'}$ in the Time attribute (or column) of $S$ to be shipped to the join region $N_F$ are the values in positions 1, 4, and 5 according to $\text{Bitmap}(S)$.

Figure 4 shows the join step of the execution of the given BEJ query. A nested-loop join is performed between $P_{R'}$ and $P_{S'}$ of the join columns of $R$ and $S$ creating two sets of position list: one for $R$ and one for $S$. Each entry pair in the position lists contains the position in $P_{R'}$ and $P_{S'}$ that contains the same join value.

The final result step is shown in Figure 5. The two position lists created in the join step are used to stitch together values from the qualified columns in $R$ and $S$ shipped to the join region $N_F$. As one example of tuple reconstruction, the values from position 1 in columns Vid and Time of table $R$ and the value from position 3 in the Time column of table $S$ are stitched to form a tuple. The stitched tuples are shipped to the query sink as the query result.

4. Performance Analysis

To test the cost-effectiveness of the join algorithm proposed in this paper, a performance analysis is made with the existing join algorithms for sensor networks. We specifically compare with existing SNJ and RFB algorithms, which are based on traditional relational databases. Both the SNJ and RFB algorithms use filtering techniques to reduce the amount of data that are shipped to the final join region. Since our proposed algorithm is based on column-oriented databases, the performance analysis conducted is to show the effectiveness of column-oriented data storage in sensor networks. Our algorithm also uses a filtering technique to reduce the amount of data that are shipped to the final join region. In addition,
we also conduct experiments to show whether the semijoin step is necessary and effective in join processing.

The various algorithms were compared in terms of the communication cost, which is the total number of bytes transmitted among various nodes to get the join result. The communication cost is the battery energy consumed associated with all data transmitted during query processing. We tested the different join algorithms in executing the BEJ query on the given schema for R and S mentioned in the previous section.

4.1. Experimental Environment. At present, no standard benchmark is available to test the performance-enhancing techniques in sensor networks. For an objective comparison and performance analysis, we used the same experiment environment used to analyze the performance of the SNJ algorithm [8].

In the experiment, 10,000 sensor nodes were created and distributed uniformly in a 100 by 100 grid. We placed each node at the center of each grid. The query sink was placed at the center of the grid. Region R and region S were placed at the bottom right and bottom left of the grid, respectively, with each region containing 800 sensor nodes each.

Table 2 shows the values of various parameters used in the performance experiments. The additional assumptions made for the experiment are as follows. The number of hops required to route a single message from the source node to the final node was assumed to use the distance and the communication radius between the two sensor nodes. In order to simplify network traffic analysis, we assumed that no failure occurs during message transmission.

We compared the join algorithms for different table sizes to determine the effect of the increase in the number of tuples in tables R and S with regard to the communication cost. We also tested the communication cost of different join algorithms for several join selectivity. Join selectivity is the fraction of the tuples in the table that satisfies the join condition. In addition, we tested whether the semijoin step in the proposed algorithm is effective or necessary.

4.2. Experimental Result. In our first experiment, we tested the effect of the join selectivity on communication cost during query processing for various sensor network algorithms. In executing the example BEJ query, the communication cost is measured by the total number of bytes transmitted to different nodes in the sensor network to process the query.

Figure 6 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 different join selectivity tested is 0.01, 0.1, 0.3, and 0.5. Join selectivity of 0.01 means that only 1% of tuples in the tables satisfies the join condition. The size of the bitmap in our proposed algorithm is identical to the bit vector in the RFB algorithm. Our proposed algorithm outperforms both the SNJ and the RFB algorithms for all different join selectivity tested. As can be seen, the communication cost of the proposed algorithm gets lower, in some cases almost 50% cost decrease, as the join selectivity gets lower. This implies that when more data is shipped to the final join region, our algorithm performs far better than the existing algorithms.

In the next experiment, we tested the effect of different table sizes on communication cost during query processing. Figure 7 shows the communication cost of the different algorithms where the table sizes to be joined are 2000, 5000, and 8000 tuples. The selectivity tested is 0.1. It can be seen that if the sizes of table R and table S to be joined increase, the communication cost of our algorithm gets lower compared to both the SNJ and the RFB algorithms. The performance of our proposed algorithm gets better as the size of tables to be joined gets bigger. The performance gain, or the decrease in communication cost, is more than 25% in some cases. Again, this implies that as more data are to be joined and shipped to the final join region, our algorithm performs better than the existing algorithms.

The aim of the next performance comparison is to measure the effectiveness of the semijoin step in our proposed algorithm. The aim of the semijoin step is to filter out unwanted data before shipping the data for final join. Figure 8 shows the result of our experiment that compares the communication cost using the proposed algorithm with or without the semijoin step for various join selectivity. The experiment assumes that the number of qualified columns during join is two columns per join table, as is the case in
our example BEJ query. It can be seen that the semijoin step lowers the communication cost for all join selectivity during query processing. The reduction in communication cost is as much as 50% during query processing.

The final experiment is to test the effect of the number of qualified columns in the join tables during join processing in measuring the communication cost. In the SNJ algorithm, based on the traditional row-oriented or relational databases, the number of qualified columns during join processing does not affect the communication cost. This is obvious as the tuples, regardless of the number of columns per tuple that participate in final join, must be shipped to the join region. In column-oriented databases, the number of qualified columns matter, since only those column values are shipped to the final join region.

Figure 9 shows the result of the experiment to test the effect of qualified columns during join processing. It is shown that when the semijoin step is performed to filter out unwanted data, the communication cost using our proposed algorithm is always lower than when using the SNJ algorithm. The saving in communication cost is almost 50%. In the proposed algorithm, the increase in communication cost doubles every time the number of qualified columns increases during join. When the number of qualified columns is 1, which means that only the join column in the join tables qualifies, the communication cost is the same regardless of the semijoin step. However, if the number of qualified columns is more than one in the join query, the communication cost is always higher than the SNJ algorithm if the semijoin step is bypassed in our proposed algorithm.

Regardless of the size of tables, join selectivity, and the number of qualified columns, our proposed algorithm that includes the semijoin step to filter out unwanted data always outperforms the existing SNJ and RFB algorithms. In addition, the semijoin step is necessary when the number of columns that qualifies the given join query is more than one column.

5. Conclusion

The main focus of the existing join strategies in sensor network databases was how to select the sensor nodes in
the join region to perform the final join. Just a few research results were reported on the use of some filtering techniques to prune unnecessary tuples before shipping the candidate tuples to the join region. The SNJ algorithm uses synopsis, which is similar to semijoin, to prune unwanted tuples. The RFB algorithm uses semijoin and bit vectors to filter out unwanted tuples. These algorithms reduce communication cost since less data is shipped to perform join and saves limited battery power. They are all based on traditional databases.

In this paper, we proposed a join algorithm for efficient processing of join queries in sensor networks based on column-oriented databases. Since most queries for sensor network are read-only with no update, physically storing the sensor data in columns, and not rows, in the nodes makes sense. The advantage of storing the sensor data in columns is that only the selected columns required for join are shipped to the join region. Together with a filtering technique based on semijoin and bitmap, we believe that the proposed algorithm is more energy efficient than similar join algorithms based on the traditional databases.

Experimental results show that our proposed join algorithm outperforms both the SNJ and the RFB algorithms in reducing the communication cost of BEJ join queries in sensor networks. The saving in communication cost is almost 50% in many cases compared to existing algorithms. To validate our experiment, we used different values of join selectivity as well as different cardinalities of tables. In addition, we also experimented with the effect of filtering and the number of qualified columns on communication cost. Through performance analysis, we showed that the performance of the proposed algorithm gets better as the join selectivity, table size, and number of qualified columns increase.

Overall, as more candidate tuples are shipped to the join region, our proposed algorithm is more energy efficient than both the SNJ and the RFB algorithms. However, if filtering technique is not performed to reduce the amount of data to be shipped to the final join region, our algorithm that excludes the semijoin step is less energy efficient than both the SNJ and RFB algorithms. In summary, we showed that using a filtering technique in column-oriented databases is more energy efficient than any existing join algorithms for sensor networks.

Acknowledgment

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Grant no. 2012-0007012).

References

[1] J. J. Kim, I. S. Shin, Y. S. Zhang, D. O. Kim, and K. J. Han, "Aggregate queries in wireless sensor networks," International Journal of Distributed Sensor Networks, vol. 2012, Article ID 625798, 15 pages, 2012.
[2] C.-H. Lee, C.-W. Chung, and S.-J. Chun, "Effective processing of continuous group-by aggregate queries in sensor networks," Journal of Systems and Software, vol. 83, no. 12, pp. 2627–2641, 2010.
[3] C.-C. Hung and W.-C. Peng, "Optimizing in-network aggregate queries in wireless sensor networks for energy saving," Data and Knowledge Engineering, vol. 70, no. 7, pp. 617–641, 2011.
[4] Q. Ren and Q. Liang, "Energy and quality aware query processing in wireless sensor database systems," Information Sciences, vol. 177, no. 10, pp. 2188–2205, 2007.
[5] S. J. Lim and M. S. Park, “Energy-efficient chain formation algorithm for data gathering in wireless sensor networks,” International Journal of Distributed Sensor Networks, vol. 2012, Article ID 843413, 9 pages, 2012.
[6] H. Gupta, Z. Zhou, S. R. Das, and Q. Gu, "Connected sensor cover: self-organization of sensor networks for efficient query execution,” IEEE/ACM Transactions on Networking, vol. 14, no. 1, pp. 55–67, 2006.
[7] A. Coman, M. A. Nascimento, and J. Sander, "On join location in sensor networks,” in Proceedings of the 8th International Conference on Mobile Data Management (MDM ‘07), pp. 190–197, Mannheim, Germany, May 2007.
[8] H. Yu, E.-P. Lim, and J. Zhang, “On in-network synopsis Join processing for sensor networks,” in Proceedings of the 7th International Conference on Mobile Data Management (MDM ’06), Nara, Japan, May 2006.
[9] K.-C. Kim and B.-J. Oh, “An energy efficient filtering approach to in-network join processing in sensor network databases,” Communications in Computer and Information Science, vol. 263, no. 2, pp. 116–122, 2011.
[10] H. C. Lee, Y. J. Lee, J. H. Lim, and D. H. Kim, “IJA: an efficient algorithm for query processing in sensor networks,” Sensors, vol. 11, no. 2, pp. 1682–1692, 2011.
[11] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden, "Materialization strategies in a column-oriented DBMS,” in Proceedings of the 23rd International Conference on Data Engineering (ICDE’07), pp. 466–475, Istanbul, Turkey, April 2007.
[12] A. Ailamaki, D. J. DeWitt, and M. D. Hill, “Data page layouts for relational databases on deep memory hierarchies,” VLDB Journal, vol. 11, no. 3, pp. 198–215, 2002.
[13] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. row-stores: how different are they really?” in Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD ’08), pp. 967–980, Vancouver, Canada, June 2008.