ISSN 2278 - 3091

International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.5, Pages :54-58 (2013)
Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India

Bloom Vector Join for Sensor Query Processing

G. M. Sridevi, T. V. Rohini, K. Kameswari and M. V. Ramakrishna
mvrama@yahoo.com
Department of ISE, SJB Institute of Technology
Kengeri, Bangalore 560 060

***Abstract*—Sensors are low-cost devices which sense and disseminate information about the environment around them. A set of such nodes which can communicate by radio transmission form a sensor network, which communicates with the outside world through a sink node. In view of their low cost and adhoc nature of deployment, sensor nodes have computational constraints and limited power supply. Each node produces a stream of sensor data, all of which may not be stored or used. The user poses queries to the sensor network to obtain the necessary information about the environment. The sensor responds by transmitting the required data. Transmission is typically energy consuming operation. The queries may require data from more than one node, such as join of data produced at two nodes. This paper deals with the specific problem of efficient join processing using data from two nodes. We investigate the use of Bloom vector to perform join operation on data streams produced at two nodes, so as to minimize the number of bytes transmitted. We propose different methods applicable based on the accessibility of the nodes and provide a cost analysis of the methods. Our results indicate that the proposed method gives a significant improvement when compared to the traditional method for normal values of join selectivity.**

***Keywords*—Bloom Vector, Data Streams**

INTRODUCTION

Sensors are devices that measure a physical quantity such as temperature and give a corresponding electrical signal. The sensor nodes communicate by wireless transmission. Sensor nodes are constrained in battery power, memory and processing capacities. The sensor nodes collect unbounded continuous data stream of values. The sensors are connected to the outside world through a Link node. The data collected by the sensors are requested by users using queries. The queries may originate from one of the nodes or from external sources and the nodes respond by sending relevant data. Certain blocking operations such as sort and symmetric join are not allowed over such streams unless the user specifies a bounded subset of stream or window [1], [2]. This paper deals with the issue of computing join efficiently between data of two different nodes. We propose the use of bloom vector in order to reduce the amount of data transmitted [3]. A Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set or not.

The rest of the paper is organized as follows. In the next section we give a brief description on sensor applications and use of bloom vector. The proposed method is presented in section 3. Section 4 provides comparision of the cost analysis between the traditional and proposed method. Conclusions and future work are presented in section 5.

BACKGROUND

Sensor networks have a variety of applications. Examples include environmental monitoring (air, soil and water, tsunami), habitat monitoring (determining the plant and animal species population and behavior), seismic detection, military surveillance, inventory tracking etc [4]. In many of these applications, data is produced continuously and hence it is called “data stream”. It is obviously not practical to store all the data produced. Storing and querying data streams pose a serious challenge for data management systems as traditional DBMS paradigm of set-oriented processing of disk-resident tuples does not apply. Join is an important operation in a data stream processing system [5]. Kang and Viglas described a “Moving Window Join” which run continuously and produce new results as new tuples arrive [6], [7], [8], [9]. Let R and S be two data streams that contain a join attribute ’A’ and ’B’. The Equi-Join of R and S is the subset of the cross product of the two streams that contains exactly those pairs of tuples(r, s) such that $r \in R$ and $s \in S$ and r.A=s.B [10].

Processing a join over unbounded input streams requires unbounded memory, since every tuple in one infinite stream must be compared with every tuple in other. This is not practical in a sensor network due to limited memory and processing capabilities. Join queries for sensor data should contain “window predicates” that

ISSN 2278 - 3091

International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.5, Pages :54-58 (2013)
Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India

TABLE 1
HASH VALUES FOR KEYS X, Y, Z

Key	h_1(key)	h_2(key)	h_3(key)
x	1	5	13
y	4	11	16
z	3	5	11

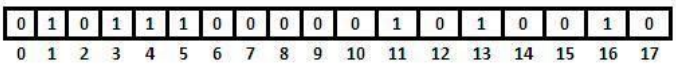

Fig. 1. Example for Sensor Network

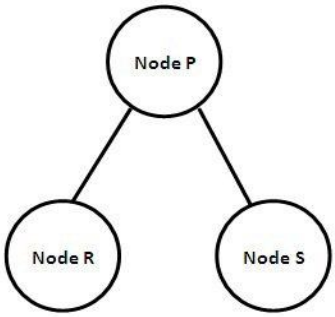

Fig. 2. Problem Scenario

restrict the number of tuples that must be stored for each stream [6]. For the above example, we might specify that we are only interested in R tuples that have arrived in last t_1 seconds and S tuples that have arrived in the last t_2 seconds. We next describe bloom vector and its use [11].

Bloom Filter

A Bloom filter is a space-efficient probabilistic data structure that is used to test whether a given element is a member of a set [12], [13], [14]. False positive retrieval results are possible, but false negatives are not; i.e a query returns either "inside set (may be wrong)" or "definitely not in set". Elements can be added to the set, but not removed. For a given vector size, as more elements are added to the set, the probability of false positives increases.

To start with, the Bloom filter is a bit array of m bits, all set to 0. There are k different hash functions, each of which maps or hashes a set element to one of the m array positions with a uniform random distribution. To add an element x, we compute $h_1(x)$, $h_2(x)$, ... $h_k(x)$ and set the corresponding bit positions to 1. To query if y is present, we compute $h_1(y)$, $h_2(y)$, ... $h_k(y)$ and check to see if the bit is set. If any of the bits at these k positions happens to be 0, then the element is definitely not in the set. If all are 1 then y is in the set with a very high probability.

Fig 1 shows an example of a Bloom filter, representing the set x, y, z. The table 1 shows the positions in the bit array that each element is mapped to. For this figure, $m = 18$ and $k = 3$ where m is bloom vector size and k is number of hash functions. For example consider the element w, which hashes to 4, 13 and 15. As the bit at address 15 is 0, we can conclude that the element w does not belong to the set. For an element v, suppose the hash values are 3, 5 and 11. All the bit positions are 1 and we will falsely conclude that v belongs to the set. It is called a false positive If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive.

BLOOM VECTOR JOIN

The scenario we consider is as follows. One of the nodes is the base station and is connected to the outside world. All queries arrive at the base station from outside world, although they could also originate from the other nodes. The base station communicates the query to the required sensor node. Depending on the query the response may be one-time or periodic. For example, if the query is "report current temperature", we have one response. If the query is "report average temperature every hour on the hour", the response will be one tuple every hour. Specifically, we consider join queries. The query involves join of the tuples being produced at node R and node S. A and B are the join attributes in relation R and S respectively, the join being on the condition R.A=S.B. The result is required at node P as shown in the Fig 2. Nodes R and S are able to communicate with node P either directly or through other intermediate nodes. P may be the base station. Nodes R and S may or may not be able to communicate directly (that is within the transmission range of each other).

A. Traditional Method

The query transmitted from node P to nodes R and S, should specify that the join involves tuples that have arrived at R in last t_1 seconds and tuples that have arrived at S in the last t_2 seconds is requested. In response to such a query, nodes R and S send all the relevant data to node P. Node P computes the join. The join result is sent to the requesting node. If there are 100 tuples each of 50 bytes at both the nodes R and S. R and S have to

ISSN 2278 - 3091

International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.5, Pages :54-58 (2013)
Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India

send 5000 bytes each of which are transmitted to node P. Totally 10000 bytes need to be transmitted. The energy consumed for data transmission is proportional to the amount of data transmitted. If the join selectivity is low, most of the tuples received at P will not participate in the result and hence the transmission energy is wasted. Our idea is to use the bloom vector technique to reduce the amount of data transmitted, by selecting only the relevant tuples for transmission.

B. Bloom Vector Join

Let us assume that there is no direct communication between nodes R and S and all the communication takes place through parent node P. Node R stores the data arrived during the last t_1 time interval and node S stores the data that arrived during the last t_2 time interval due to memory limitations. We assume that the nodes communicate about the hash functions beforehand. We use 3 hash functions $h_1(key)$, $h_2(key)$ and $h_3(key)$.

- When a query arrives at node R, it generates the bloom vector BV_R by computing the 3 hash addresses for each element of the attribute 'A' of each tuple. The bloom vector size depends on the number of tuples in the relation R. For m tuples, we suggest a bloom vector of m bytes, i.e m 8 bits. Thus, the load factor will be 3/8 * 100 = 37.5% which is adequate to give a very low false positive rate [11].
- The bloom vector BV_R is sent to node P.
- Parent P forwards the bloom vector BV_R to node S.

  At node S the same 3 hash functions $h_1(key)$, $h_2(key)$ and $h_3(key)$ are used on the tuples in S, by passing each element of attribute 'B' as the key to compute the addresses. For each key, we check to see if the addresses generated from the hash functions are set in the bloom vector of R. If all the three positions in the bloom vector are set to 1 then, we can conclude that the corresponding matching key might be present at node R and the tuple in S will be selected.

  Bloom vector BV_S is generated only for the selected tuples at node S (Note: The bloom vector size can be smaller depending on the number of selected tuples in S).
- BV_S and the selected tuples are forwarded from node S to parent P, then to R.
- Node R matches its tuples with BV_S similar to the above.
- Node R sends the matching tuples to P.

0	1	1	1	0	1	1	1	1	0	0	1	0	1	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Fig. 3. Bloom Vector BV_R Generated by Node R

- P performs the Equi-Join of the tuples from nodes R and S and forwards the result to the sink node. Note that any false positive tuples are taken care of automatically during the join operation.

We illustrate the above algorithm with a small example, with relation R having 5 tuples with values of attribute 'A' as 8, 13, 15, 20, 25. The relation S has 6 tuples with values of attribute 'B' as 5, 7, 8, 12, 15, 30. The adequate bloom vector size for R would be 5*8=40 bit positions. We use a bloom vector size of 19 bits only, to keep the diagram small. Initially all the bits are set to 0.

We use the hash functions,

$$h_1(x) = x \bmod 19$$
$$h_2(x) = 3x \bmod 19$$
$$h_3(x) = 5x \bmod 19$$

At node R, the hash values are computed for each element of attribute A . The resulant hash positions for the key 8 is $h_1(8)=8$, $h_2(8)=5$ and $h_3(8)=2$. Then the bit positions 8, 5 and 2 are set to 1 in the bloom vector BV_R . Similarly, R computes the hash functions on remaining keys, i.e 13, 15, 20, 25 and the corresponding bits in the bloom vector are set to 1. Fig 3 shows the bloom vector after all bits have been set.

The bloom vector BV_R generated at node R is forwarded to node S through the parent node P. At node S, the above hash functions are applied on attribute B for the tuples in S and the resultant addresses are checked with the bloom vector of R. If all the three hash values are set to 1 in the bloom vector of R, then that tuple will be selected. The value 8 generates hash address 8, 5, 2. All the three values are matching with bloom vector of R. Hence the corresponding tuple is selected. If any one of the hash positions is 0 in the bloom vector, then that tuple will not be selected.

There is a possibility of some false keys being selected. Consider the key 5 in node S which hashes to positions 5, 15, 6 which are all set in bloom vector BV_R. But the key 5 does not exist in set R. Therefore, this is a false positive. But it does not cause any problem and the only consequence is that an extra tuple is sent to node P. For the selected tuples in S, bloom vector BV_s is generated and it is forwarded to R through P. Also the

ISSN 2278 - 3091

International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.5, Pages :54-58 (2013)
Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India

selected tuples in S are sent to P. Similar comparision is done in R to get the tuples from node R and then the selected tuples are sent to node P. Node P now contains all the matching tuples from R and S and P computes the join between the selected tuples. False key tuples are eliminated during the join operation in the parent P.

C. Bloom Vector Join with Direct communication

Consider the case when there is direct communication between nodes R and S, i.e data and results can be exchanged directly between R and S without going through P. When the join query is sent from parent P to nodes R and S, we compute the join as follows.

- R computes the bloom vector for the tuples using attribute 'A' and transmits it to node S.
- S uses the bloom vector received from R. It selects the matching tuples by hashing the data in attribute B(similar to indirect communication) and transmits the tuples to P.
- S also sends a bloom vector(only for the selected tuples in node S) to node R
- Node R selects the matching tuples using the received bloom vector of S and attribute 'A'and sends the matching tuples to P
- P computes the join using the tuples received from R and S.

The same can be accomplished by sending the bloom vector of S to R first.

We also considered the following alternative. R sends its bloom vector to S and S computes the selected tuples and transmits them to R. R then computes the join using the tuples received for S and the join result is transmitted to P. The cost with this method is marginally more than the above method. Hence we will not elaborate further on this method.

COST ANALYSIS

We provide a cost analysis of the different methods. The number of bytes transmitted is the cost measure. Power consumed is a direct funtion of the number of bytes transmitted. We ignore the overheads of transmission such as preamble, etc in the packets and consider the payload bytes only. We assume the bloom vector size for node R as n_r bytes(n_r*8 bits).

We use the following notations

Number of tuples in Relation R = n_r
Number of bytes in each tuple in R = b_r
Number of tuples in Relation S = n_s
Number of bytes in each tuple in S = b_s

Selectivity at node S, i.e the relative number of tuples relevant for the join = σ_s

From among the n_s tuples at S, $n_s\sigma_s$ tuples will be participating in the join operation. Note that this assumes ideal performance of the bloom vector and hence does not consider the effect of the false positives. However, this error due to false positives is insignificant.

A. Traditional Method

Number of bytes transferred in the straight forward approach can be shown as follows,

Tuples from Node R to Node P : $n_r b_r$
Tuples from Node S to node P : $n_s b_s$
Total Cost = $n_r b_r + n_s b_s$

B. Bloom Vector Method with Indirect Communication

The number of bytes transferred for this scenario can be shown as follows.

- Bloom Vector from Node R to Node P : n_r (size of the bloom vector of R)
- Bloom Vector from Node P to Node S : n_r (size of the bloom vector of R)
- Selected tuples from Node S to node P : $b_s n_s \sigma_s$
- Bloom Vector from Node S to Node P and from P to R : $2n_s\sigma_s$ (size of the bloom vector of selected tuples of S)
- Selected tuples from Node R to node P : $b_r n_r \sigma_r$

Join will be performed at P.

Total Cost = $2n_r + b_s n_s \sigma_s + 2n_s\sigma_s + b_r n_r \sigma_r$

C. Bloom Vector Method with Direct Communication

Let us consider the first scenario. The number of bytes transferred for this scenario can be shown as follows

- Bloom Vector from Node R to Node S : n_r (size of the bloom vector of R)
- Selected tuples from Node S to node P : $b_s n_s \sigma_s$
- Bloom Vector of selected tuples in S to node R : $n_s\sigma_s$
- Selected tuples from Node R to node P : $b_r n_r \sigma_r$

Total Cost = $n_r + b_s n_s \sigma_s + n_s\sigma_s + b_r n_r \sigma_r$

To give us a clear picture of cost involved, we use a numerical example. First we assume $n_r = n_s =$ 100 tuples $b_r = b_s = 50$ bytes The join selectivity factor σ varies from 0.01 to 1. We have considered $\sigma_r = \sigma_s$ for this example. The results are shown in the Table 2 with $n_r = n_s = 100$ tuples, $b_r = b_s = 50$ bytes. We observe that the number of bytes transmitted is reduced dramatically with Bloom Vector Method as compared to traditional Join Method. Table 3 shows the computed results with parameters differing between

ISSN 2278 - 3091

International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.5, Pages :54-58 (2013)
Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India

TABLE 2
NO.OF BYTES TRANSMITTED, $n_r = n_s = 100, b_r = b_s = 50$

Selectivity	Traditional	Bloom Direct	Bloom Indirect
0.01	10000	201	302
0.05	10000	650	710
0.1	10000	1110	1220
0.5	10000	5150	5300
1	10000	10200	10400

TABLE 3
NO.OF BYTES TRANSMITTED,
$n_r = 100, n_s = 1000, b_r = 20, b_s = 100$

Selectivity	Traditional	Bloom Direct	Bloom Indirect
0.01	102000	1130	1240
0.05	102000	5250	5400
0.1	102000	10400	10600
0.5	102000	51600	52200
1	102000	103100	104200

the two tables with $n_r = 100, n_s = 1000, b_r = 20, b_s = 100$. The benefit of bloom vector join is even more significant in this case. For example, the number of bytes transferred is only one percent of the traditional join method when the selectivity is 0.01.

CONCLUSIONS

In this paper we addressed the issue of reducing data transfer while processing join queries in wireless sensor networks. We proposed a new method based on Bloom vector, and provided an analysis of the approach in comparison to the traditional method. Based on the results of the analysis, we conclude that for small values of join selectivity, the Bloom vector based method reduces the amount of data transfer quite significantly.

REFERENCES

[1] J. Gehrke and S. Madden, "Query processing in sensor networks," *Pervasive Computing, IEEE*, vol. 3, no. 1, pp. 46–55, 2004.

[2] L. Golab, S. Garg, and M. T. Özsu, "On indexing sliding windows over online data streams," in *Advances in Database Technology-EDBT 2004*. Springer, 2004, pp. 712–729.

[3] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 5, pp. 604–612, 2002.

[4] A. Bharathidasan and V. A. S. Ponduru, "Sensor networks: An overview."

[5] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs, "Algorithms and metrics for processing multiple heterogeneous continuous queries," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 1, p. 5, 2008.

[6] J. Kang, J. F. Naughton, and S. D. Viglas, "Evaluating window joins over unbounded streams," in *Data Engineering, 2003. Proceedings. 19th International Conference on*. IEEE, 2003, pp. 341–352.

[7] H.-S. Lim, J.-G. Lee, M.-J. Lee, K.-Y. Whang, and I.-Y. Song, "Continuous query processing in data streams using duality of data and queries," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 313–324.

[8] J. Krämer and B. Seeger, "Semantics and implementation of continuous sliding window queries over data streams," *ACM Transactions on Database Systems (TODS)*, vol. 34, no. 1, p. 4, 2009.

[9] J. Teubner and R. Mueller, "How soccer players would do stream joins," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 625–636.

[10] A. Das, J. Gehrke, and M. Riedewald, "Approximate join processing over data streams," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 40–51.

[11] M. V. Ramakrishna, "Practical performance of bloom filters and parallel free-text searching," *Commun. ACM*, vol. 32, no. 10, pp. 1237–1239, Oct. 1989.

[12] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: building a better bloom filter," in *Algorithms–ESA 2006*. Springer, 2006, pp. 456–467.

[13] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, "On the false-positive rate of bloom filters," *Information Processing Letters*, vol. 108, no. 4, pp. 210–213, 2008.

[14] M. Zhong, P. Lu, K. Shen, and J. Seiferas, "Optimizing data popularity conscious bloom filters," in *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*. ACM, 2008, pp. 355–364.