IEEE-International Conference on Recent Trends in Information Technology, ICRTIT 2011
MIT, Anna University, Chennai. June 3-5, 2011

Distributed Database Management and Join of Multiple Data Streams in Wireless Sensor Network using Querying Techniques

Vaidehi V[#1], Sharmila Devi D[*2]

[# *] Department of Information Technology, Madras Institute of Technology
Chennai, India

[1] vaidehi@mitindia.edu

[2] sharmiladevi.it50@gmail.com

***Abstract*—Sensor networks are multi hop wireless networks formed by a large number of resource-constrained sensor nodes. The events detected by the sensor generate a stream of data. Centralized join query processing algorithm incur more communication overhead due to frequent exchange of data between the sink and the sensor nodes. To query or access data generated by the sensor nodes, the sensor network can be viewed as a distributed database. Communication-efficient implementation for join of multiple data streams in a sensor network is particularly challenging due to unique characteristics of the sensor networks such as limited memory and battery energy on individual nodes. Hence the design of Distributed Nested Loop Join Processing (DNLJP) algorithm has been proposed in this paper. The proposed scheme groups sensors based on geographic locations to form a cluster and perform the query processing in a distributed manner over the data collected across different regions. DNLJP also optimizes the query based on the desired optimization criteria like query cost, query execution time etc., and applies the corresponding query processing technique to achieve the desired result. Analysis shows that the communication overhead of the proposed distributed algorithm is reasonably low and the efficiency of the query processing is considerably improved over a wide range of query.**

***Keywords*—Sensor networks, Query optimization, Query processing, select-project-join databases.**

I. INTRODUCTION

From a data storage point of view, a sensor network is viewed as a distributed database that collects physical measurements about the environment indexes them, and then serves queries from users. Each sensor node typically generates a stream of data items that are obtained from the sensing devices on the node [6]. Generally, each sensor consists of a small node with sensing, computing, and communication capabilities.

The event (data) is transmitted through a multihop communication route to a centralized sensor node (i.e., sink node), which needs more energy.

Like a database, the sensor network can be queried, and efficient in-network implementation of database queries is of great importance. It is advantageous to express queries to a sensor network database at a logical, declarative level, using relational languages such as SQL.

Join is an important database operator, which forms the basis of a deductive query engine for sensor networks. Join operator can be used to represent complex events in sensor network. Thus efficient implementation of join in sensor networks is of great importance due to limited network resources.

Popular distributed implementation for join in sensor network is Perpendicular Approach (PA) [3]. PA is communication efficient, load balanced and incurs near optimal communication cost for binary joins under the assumption of uniform generation of tuples across the network. PA works by using appropriately defined horizontal and vertical paths for tuple storage and join computation respectively. However this scheme is computational heavy due to large intermediate results.

Centralised approach is a popular approach [2], [3], [7]. In this approach, join of data streams is performed by central server on the tuples generated by nodes. However, schemes without in-network processing may incur prohibitive communication costs. Hence a better scheme has to be devised suitable for WSN.

Design of communication-efficient and load-balanced in-network implementations of join in sensor networks is particularly challenging due to limited memory available at each node and arbitrary network topologies [6]. Hence the design of Distributed Nested Loop Join Processing (DNLJP) algorithm has been proposed in this paper. The proposed

978-1-4577-0590-8/11/$26.00 ©2011 IEEE

scheme groups sensors based on geographic locations to form a cluster and perform the query processing in a distributed manner over the data collected across different regions.

This paper is organised as follows: Section II explains the Proposed scheme, Section III presents the Results.

II. PROPOSED SCHEME

A. Distributed Nested Loop Join Processing Algorithm (DNLJP)

The Distributed Nested Loop join algorithm is a simple nested loop algorithm that joins two relations X and Y by making nested loops.

For each tuple x in X do
For each tuple y in Y do
If x and y satisfy the join condition
Then output the tuple <x, y>

The DNLJP uses one join input as the outer input table and one as the inner input table. The outer loop consumes the outer input table row by row. The inner loop, executed for each outer row, searches for matching rows in the inner input table.

B. Procedure

Step 1: *Forming the coordinator area for the join region R_J.* Designate the nodes closest to the centres C_A and C_B of the regions R_A, R_B as regional coordinators respectively (Fig 1). Designate the coordinator location C_J for join region R_J.

Step 2: The Routing trees (In Fig 2) are established in regions R_A and R_B rooted at their respective coordinator nodes C_A and C_B. ops_A, ops_B indicate the operation of region A and B respectively.

Step 3: The coordinators of each region gather tuples required for processing the query. Each coordinator sends this information to C_J of the join region R_J.

Step 4: Construct the join region: C_J constructs R_J so that it has sufficient memory space at its nodes to buffer A.

Step 5: *Distribute A over R_J:*
(a) C_J asks C_A to start sending packets with tuples. Once C_J receives *A*'s tuples, it forwards them to a node in R_J with available memory.
(b) Upon receiving a request for data from C_J, C_A asks for relevant tuples from its children in the routing tree. The process is repeated by all internal tree nodes until all relevant tuples have been forwarded up in the tree.

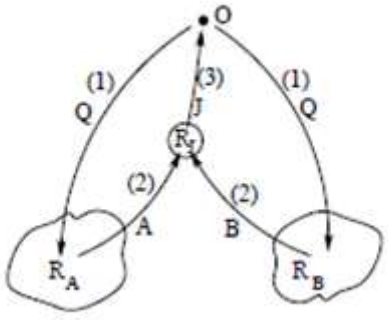

Fig 1: Join operation between two regions

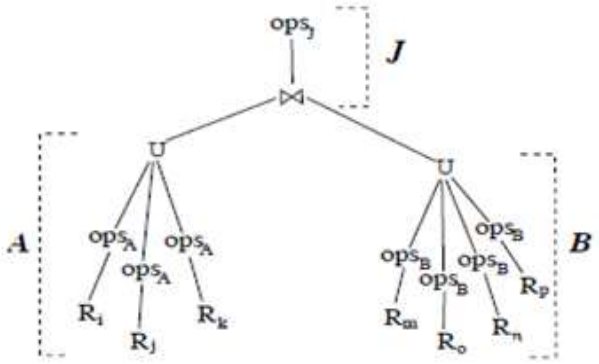

Fig 2 Join region

Step 6: *Broadcast B over R_J:*
(a) Upon receiving a request for tuples from C_J, C_B asks for a *number* of join tuples from its children in the routing tree. The process is repeated by all internal tree nodes if they cannot satisfy the request alone.
(b) Once C_J receives requests for *B*'s tuples from all nodes in R_J, Step 6 is repeated unless C_B signals that it has no more packets (i.e., tuples) to send.

C. Efficient Query Processing for Tracing RFID Tags

The performance of the above query joining technique in distributed database management has been validated using RFID sensor deployed in an organization for checking the employee profile.

For efficient query processing of tracing RFID tags [2][3], an index structure is used based on RFID tag events generated when a RFID tag goes in and out of a location where a RFID reader is placed. From the time stamped historical information contained in RFID tag events, RFID tracing application uses Location Identifier (LID), Sensor identifier (SID), RFID Tag Identifier (TID), and the identified time (TIME) as predicates for tracking and tracing RFID tags [4].

The information gathered by RFID Tag tracing system is represented as EPCIS (Electronic Product Code Information Services) tag events and stored in the persistent storage in order to answer tag related queries [4][5]. Since the tag event contains several elements of time stamped historical information, it can represent the dynamic flow of tagged items between RFID locations along tag routes. If an RFID application needs a history of these items, a query processor can respond to the application by retrieving suitable tag events in a repository.

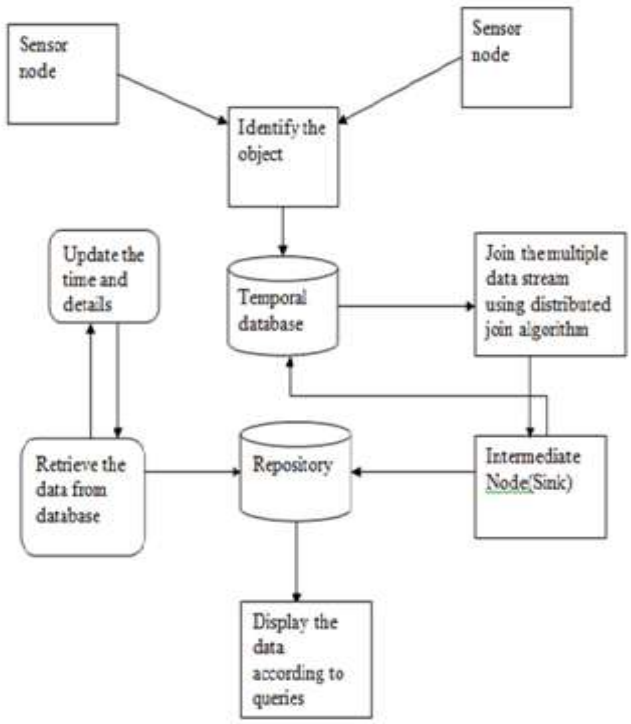

Fig 3: Architecture Diagram

Fig 3 shows the architecture diagram in which each and every sensor node has its own database, to query or access data generated by the sensor nodes. It minimizes the communication cost because it is an in-network implementation strategy. Traffic is reduced due to low communication between sensors to sink.

D. Distributed Database Query Optimization Analysis

Query optimizer for relational databases use relational algebra (RA) for internal query representation. The goal of optimization is to reduce the execution cost of a set of queries by performing their common tasks. A query may have more than one solution plan [9]. Query optimization aims to minimize the time required to calculate the output for a given query. In Fig 4 we assume that queries are expressed in terms of some uniform language and a query processor will generate a query plan[8] [10] to answer an input query by decomposing an input query into sub queries to relevant information sources and determine a correct and efficient order to execute these sub queries.

For example, an implementation program with a minimum total cost may be found, and then the program should be amended under the conditions of without increasing the total cost to achieve the shortest response time. In some cases, the optimization objectives of the query processing are to reduce communication costs and at the same time to reduce response time.

Cache Replacement Policy:

When more number of events occur, the distributed database size may not be adequate to store the events. Hence using MRU (Most recently used) record is deleted from database after responding to the query to the sink through cluster head. New events are appended to the sensor database. In spite of deleting the records, if the node's memory is not adequate, then the memory blocks are periodically sent to the cluster head based on FIFO policy to save the new events.

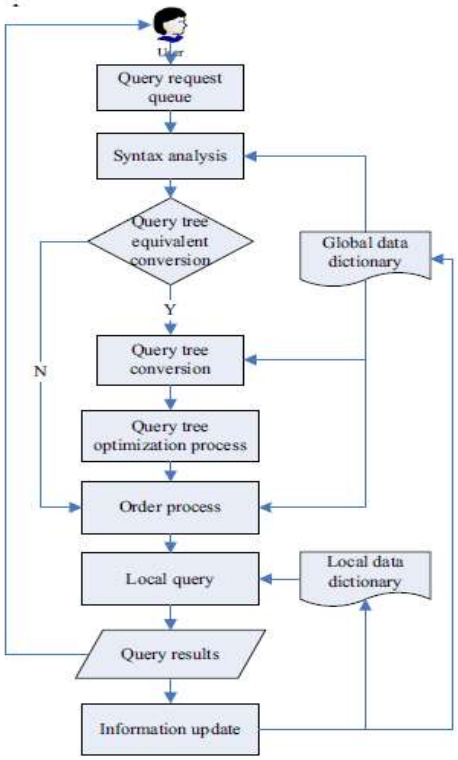

Fig 4: Distributed database query optimization process flow

Let the Ri1, Ri2, …. Rin represent the number of entries (Records) cached in the memory of the sensor i. And Tf represent the flush timer (Tf) denoting the time interval in which the joined records need to be flushed out of the sensor memory. Fig 5 presents the algorithm for cache replacement.

E. Queries and Relational Algebra Analysis

RFID Data Query Processing in a SQL-Based Stream Query Language:

RFID tag data has unique ID and time stamped. Some of the common RFID data processing tasks are very well-suited for the SQL-based stream query language presented in [8] [14]. Some common tasks in RFID data processing include duplicate elimination, context retrieval for tag IDs, and

database updates and data aggregation. All these tasks are suitable for SQL-based query languages.

.

Duplicate Elimination

Duplications can be caused by duplicated tags, duplicated readers, or simply repeated reads on the same RFID tag [12]. To eliminate the duplicates, identical tag reading appearing between the threshold time limit (say 1 second) will be considered as the single entry in the database.

1. Start the timer T_f
2. The RFID sensors S_1…..S_S perform the sensing operation of the employee sign-in and sign-out.
3. Record the sensed data as records $R_{i1}, R_{i2}, \ldots, R_{in}$ (i ϵ {1…S}) in their respective databases.
4. When the timer T_f expires do

   begin
   a. Perform Join on the records $R_{i1}, R_{i2}, \ldots, R_{in}$ (i ϵ {1…S})
   b. Store the result on the server/region coordinator performing the join operation on the records.
   c. Flush those records in the basis of MRU that have been joined completely from the memory of the sensor S_i using DELETE query, leaving behind the un-joined records in the sensor memory and thereby creating more room for additional records.

   End
5. Go to step 1

Fig 5: Algorithm for Cache Replacement

Example 1: Duplicate Filtering with Join

```
INSERT INTO sensor1
SELECT * FROM SENSOR1 AS S1
WHERE NOT EXISTS
(SELECT * FROM TABLE (S1 OVER (RANGE 1 seconds
PRECEDING CURRENT)) AS S2
WHERE S1.EMP_ID = S2.EMP_ID
AND S2.OUT_TIME IS NULL;
```

The corresponding relational algebra for the above query is written as,

Π_{all} (S1, |X| empid <-(

Π empid (s2, |x| time (

σ_{intime} <- preceding 30 min, $\sigma_{outtime}$ <- null)))

TABLE I
COMBINED DATA FORMAT

TID	SID	DATE	TIME		ACTION
			Start	End	
VE0040100003B1D44	F#002	22/02/2011	09:34:45	NULL	IN
VE00401000012F9F8	E#013	22/02/2011	09:36:20	NULL	IN
BD00401000003B1D4B	F#002	22/02/2011	10:10:32	NULL	IN
VE0040100003B1D44	F#002	22/02/2011	NULL	10:50:25	OUT
BC001801096EEAA1F	C#008	22/02/2011	10:52:02	NULL	IN
BD00401000003B1D4B	F#002	22/02/2011	NULL	11:15:23	OUT

Table I shows the combined data format obtained in the cluster head.

Data Aggregation

Data Aggregation is needed for various RFID data processing tasks. For example, we may need to count the number of persons passing at a particular time stamp. Simple built-in aggregation operators, such as COUNT, are provided by most SQL-based query languages.

Example 2 Tag based aggregation

```
SELECT COUNT (EMP_ID) FROM S1
WHERE TAG_ID LIKE '10.%.%'
AND EXTRACT SERIAL (TAG_ID) > 1000
AND EXTRACT SERIAL (TAG_ID) < 5000;
```

The corresponding relational algebra for the above query is written as,

```
Πcount(empid) (σ tag <- '10%',tag <- >1000 and <5000)
```

It enables aggregation at various levels, the above shows the tag level aggregation which yields number of employees who are associated between the tag serial lying between 1000 and 5000.

Exception SEQ

List of exception sequence can be created which will arise when it satisfies the exception condition. The best example that can be stated is, when a shift employee tries to login in another shift timing it must alert stating “non shift timing”. These kinds of exception alert helps to monitor in effective way.

Example 3: Exception SEQ

```
SELECT S1.EMP_ID, S2.EMP_ID, S3.EMP_ID
FROM S1, S2, S3
WHERE EXCEPTION SEQ (S1, S2, S3)
OVER [1 HOURS FOLLOWING S1];
```

Location Tracking

To trace the locations visited by a person, a query is posted by sink using the RFID tag of the person. Based on the query, the sensor nodes check its database for the availability of the given RFID and send the reply to cluster head. The cluster head joins the reply received from several sensors and sent it to the sink.

Example 4: Location Tracking

```
INSERT INTO SENSOR1
SELECT EMP_ID, TAG_ID, IN_TIME, OUT_TIME
FROM  SENSOR_REFERENCES
```

III. RESULTS

.The RFID Tag based tracing results in sink is shown in Fig.6. Each Tag is associated with a Name and address. Fig 7 shows the snap shot of Time manager of a single sensor node using this data the time spent by a person in specific location can be obtained.

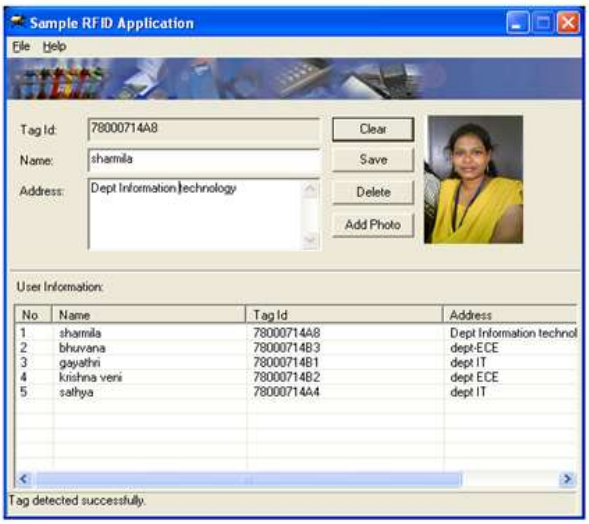

Fig 6: Tag detection

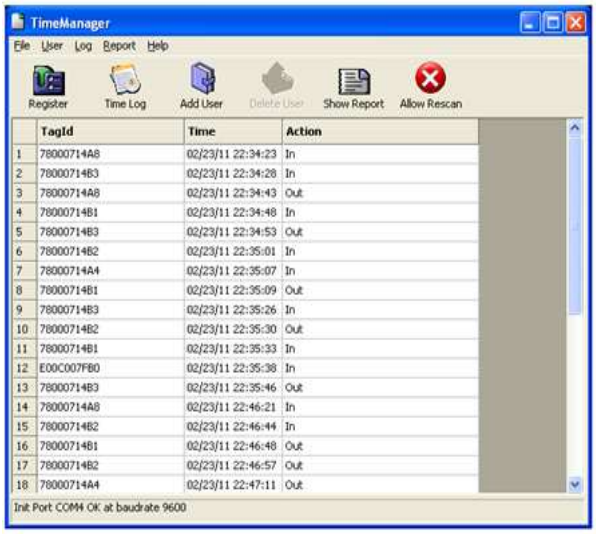

Fig 7: Time Manager

Fig 8 shows the query cost of the centralised scheme and the proposed scheme. It can be seen that the query cost is low in the proposed scheme.

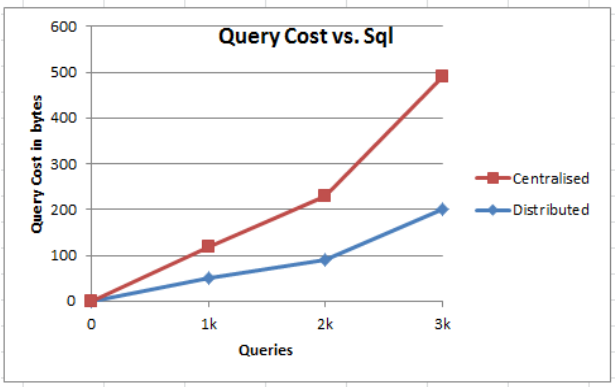

Fig 8: Query cost

Fig 9 shows the execution time taken by the centralised scheme and distributed scheme. It can be seen that the distributed scheme takes lesser execution time than centralised scheme.

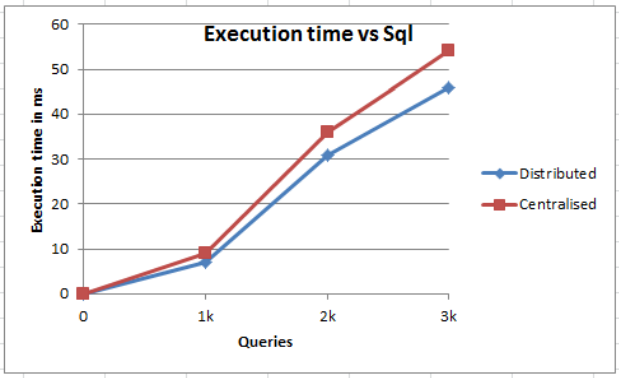

Fig 9: Execution Time

Complexity Analysis:

Join cost = Cost of accessing Table_A + # of qualifying rows in Table_A × Blocks of Table_B to be scanned for each qualifying row.

Time Complexity (TC) =outer loop iteration*inner loop iteration

The number of iterations in the inner loop is $[\log_2 n]$. The inner loop is controlled by an outer loop which is executed n times.

$$TC=>n*[\log_2 n] = [n\log_2 n]$$

IV. CONCLUSION AND FUTURE WORK

In this paper, RFID based Event detection and query optimization for distributed network has been presented. Enabling techniques such as Distributed database query optimization analysis, DNLJP algorithm, RFID Data Query Processing in a SQL-Based Stream Query Language have been proposed to reduce the query response time.

Experiments have been conducted to compare both existing and proposed approaches. The complexity of this algorithm is $O(n\log_2 n)$ is efficient than the Centralised algorithm. Analysis also show that the communication overhead of the distributed algorithm is reasonably low, which would also result in an efficient use of battery energy of the sensor nodes and the efficiency of the query processing is considerably improved over a wide range of query and network parameters.

REFERENCES

[1] R. Avnur and J. M. Hellerstein. Eddies: continuously adaptive query processing. In SIGMOD '00, 2009.

[2] A. Arasu, S. Babu and J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution," The VLDB Journal, Vol. 15 Number 2, Springer Berlin/Heidelberg, July 2005, pp. 121.-142.

[3] Y. Yao and J. Gehrke, "Query Processing for Sensor Networks," in Proceedings of Conference on Innovative Data System. Res, 2003, pp. 233-244.

[4] S. Piramuthu, "Protocols for RFID tag/reader authentication," Decision Support Systems, vol. 43, pp. 897- 914, 2007

[5] W.H.Tok and S. Bressan, "efficient adaptive processing of multiple continuous queries," in Proceedings of the International Conference On Extending Database Technology (EDBT), Prague, Italy, LNCS 2287, 2008, pp. 215-232.

[6] Fusheng Wang, Shaorong Liu, Peiya Liu, and Yijian Bai, "Bridging Physical and Virtual Worlds: Complex Event Processing for RFID Data Stream.

[7] Sharama Chakravarthy, Deepak Mishra, "An expressive event specification language for active database", Data & Knowledge Engineering archive, 2004.

[8] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In VLDB, pages 606-617,2004.

[9] Y. Ahmad and U. Cetintemel, "Network-Aware Query Processing for Stream-Based Applications," Proc. 30th Int'l Conf. Very Large Data Bases (VLDB), 2004.

[10] S.K. Gadia and S.S. Nair, Algebraic Identities and Query Optimization in a Parametric Model for Relational Temporal Databases,IEEE Trans. Knowledge and Data Eng., vol. 10, no. 5, pp. 793-807, 2008.

[11] Jin Xingyi, Lee Xiaodong, Kong Ning, Yan Baoping, "Efficient Complex Event Processing over RFID Data Stream", ICIS, 2008.

[12] J. Grant, W. Litwin, N. Roussopoulos, and T. Sellis, "Query Languages for Relational Multidatabases," VLDB J., vol. 2, no. 2, Apr. 2003.

[13] H.W. Beck, S.K. Gala, and S.B. Navathe, "Classification as a Query Processing Technique in the CANDIDE Semantic Data Model," Proc. Int'l Conf. Data Eng., pp. 572-581, 1999.

[14] Y.-N. Law, H. Wang, and C. Zaniolo, "Query Languages and Data Models for Database Sequences and Data Streams," Proc. 30th Int'l Conf. Very Large Databases (VLDB 04), VLDB Endowment, 2004, pp. 492–503.