Special Issue Published in International Journal of Trend in Research and Development (IJTRD), ISSN: 2394-9333, www.ijtrd.com

Data Stream Processing and Optimization Techniques Survey in WSN

[1]Mrs. J.Srimathi and [2]Dr.V.Valli Mayil,
[1]AP(SG)/MCA, P.hd., Research Scholar, Bharathiar Univesity, TamilNadu, India
[2]Head & Associate Professor, Dept. of Computer Science & Applications, Thanjavur, TamilNadu, India

Abstract: **WSN** is an fast growing technology and used in potential applications including Traffic Monitoring, Military applications, robotic etc. In Wireless Sensor Networks (WSNs), energy Conservation is the most important factor because each sensor devices operated with Battery. Such Wireless networks have resource constraints on data communication, computation, and energy Utilization. It is important to join and query the readings generated by group of sensors, because recovering and joining data from thousands of nodes is monotonous and impractical.

Sensor nodes needs to process data in such a way that it can generate useful information on spending only affordable amount of energy. Sensors may respond uniquely from conventional Database System. Wireless Sensor Network consists of nodes which are distributed widely. These sensors are data source for data stream processing in many cases. In this paper we discuss on stream based query processing techniques used in DSMS and evaluate continuous queries over data streams.

I. INTRODUCTION

Sensor network nodes are Stream Processing Systems (SPSs) with limited capacities. Sensor data preprocessing and query processing over data streams have been the subject of expanding consideration today. Initially, sensors regularly convey data in streams: they create data consistently, frequently at all around characterized time interims, without Querying. The processing of sensor data is that sensors are dissimilar from the well architectural data-sources usual in a merchandise DBMS. They do not provide data at consistent rates, the data is regularly jumbled, and they have inadequate processor and battery assets that the query engine needs to save at whatever point conceivable.

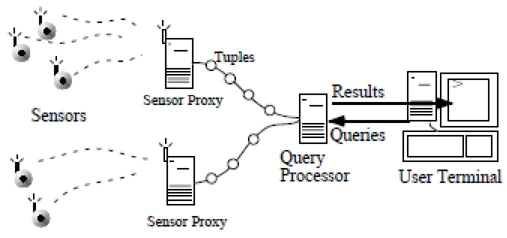

Figure 1: Environment of Sensor Query processing

DBMS: Data Base Management is a One-time Query Execution Model which store incoming tuples and submit one-time query. Query Processing is done on already stored data and retrieve the result.

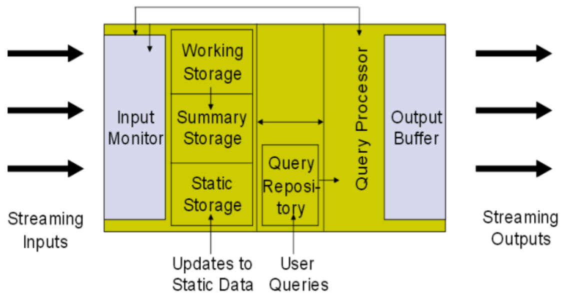

Figure 2: Overview of DSMS Architecture

DSMS: Data Stream Management is a Technique which submit continuous query from incoming streams. The input stream is processed on the fly basis and produces the results continuously to the Client. DSMS support online analysis of rapidly changing data stream.

DataStream: DataStream means real-time, continuous, ordered sequence of items, too large to store entirely not ending.

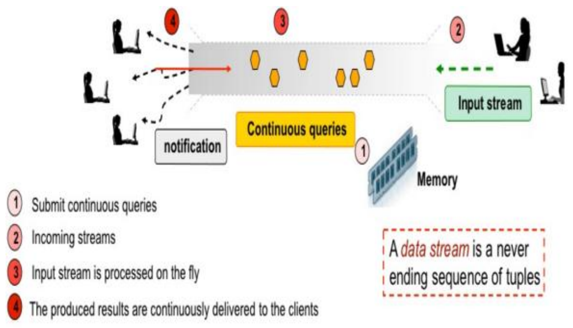
Input stream
Continuous queries
notification
Memory
1 Submit continuous queries
2 Incoming streams
3 Input stream is processed on the fly
4 The produced results are continuously delivered to the clients
A data stream is a never ending sequence of tuples

Figure 3: DataStream Management

One Time Query is a query evaluated once over the already stored tuples and Continuous query is a query which waits for future incoming tuples and evaluated continuously as new tuples arrive.

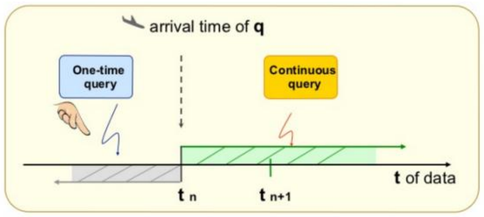

Figure 4: One-time Vs Continous Query

II NEED FOR SENSOR QUERY PROCESSING

Limitations of Sensors: Constrained resource is a vital characteristic of sensors. Inadequate resources include battery limit, communication transfer speed, and CPU cycles.

Data Streaming: The feature of sensors is that they deliver endless floods of data (at least, the sensors come up to the shortage on battery power!). Any sensor query handling framework should have the capacity to work specifically on that data streams. Since streams are limitless, administrators can never process over a whole streaming relation i.e. they cannot be *blocking*. Numerous classical operators, such as sorts, aggregates, and join algorithms therefore, cannot be utilized. Rather, the query processor must incorporate uncommon operators which convey results incrementally, handling streaming tuples each one at a time or in small blocks.

Multiple Query Processing: The query processing must be capable of monitoring continuous data and performs the operations incrementally. It facilitates the updated results of streaming operations.

III. FRAMEWORK OF CONTINUOUS QUERY PROCESSING

Query is a declarative statement requesting a subset of data. The query processor converts declarative queries into flow of data operators which is called as query plan. Some of the relational operators in queries are project, select, join, scans data from base relations, indices etc.

Query Optimizer means from the given declarative query, build the best query plan by choosing operators, orders and where to run queries. In wireless sensor, we want to combine and aggregate data streaming from motes. The current issues related to sensor database are unreliable data(come from on and offline, variable bandwidth),Push and stream based data, limited memory, power and bandwidth.

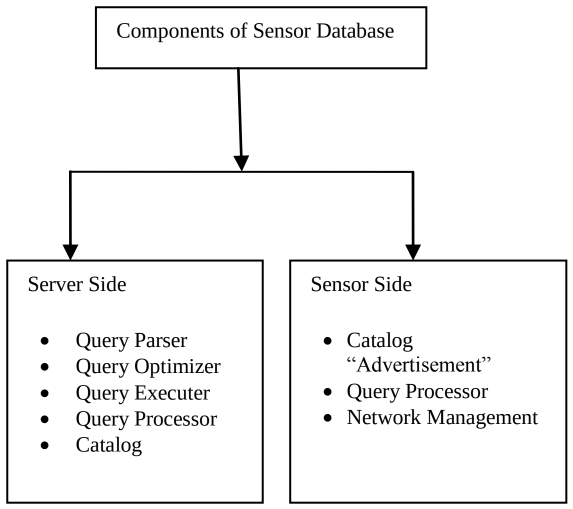

Figure 5: Components of Sensor Database

Continuous Queries:

Continuous queries are queries that are issued one after another and run continuously over the database. Continuous queries are particularly valuable in a domain like the Internet involved a lot of often evolving data. Continuous queries are persevering queries that permit clients to get new results when they get to be accessible. While continuous query framework changes an inactive web into a dynamic, they need to be able to upkeep millions of queries due to the size of the Internet.

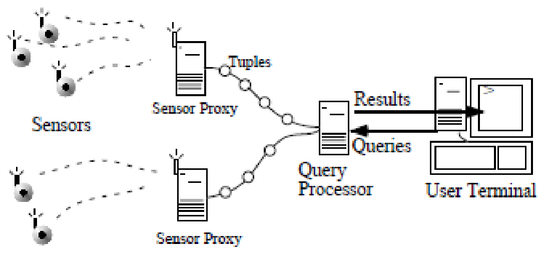

Figure 6: Continuous Query Processing

IV. SERVER SIDE SOLUTIONS FOR DATA STEAM PROCESSING IN WIRELESS SENSOR

A. Fjord Query Processing

Fjord is a query plan abstraction to handle lack of reliability and streaming, push based data. This system combines push and pull arbitrary combinations. Fjords, conversely, provide provision for **integrating streaming data** that is pushed into the system with disk-based data which is pulled by conventional operators. Fjords additionally permit to consolidate various queries into a solitary arrangement and expressly handle operators with numerous data sources and yields.

A fjord allows the query processors for the tolerant property for handling irregular data streams. Data stream system waits for the continuous flow of sensor data only when sensor tuples are *pushe*. A Fjord is a advanced query information structure which facilitates the process streaming information with conventional, disk- based information sources. The vital importance of Fjords is that they allow distributed query plans which uses **push and pull connections** between operators.

Operators and Queues used in Fjord:

Queues:

A queue is a system which manages the data from one operator to another. Queues have only one input and one output and it does not change the data it carry.

Push or pull operator is executed by the queue that connects a couple of operators: a push queue depends on its input operator to data. A pull queue requests that the input operator produce data in light of an approach to the part of the output operator. Queues implement Push and Pull pattern. Fjords additionally guide parallelism between operators by queues, state machines and OS.

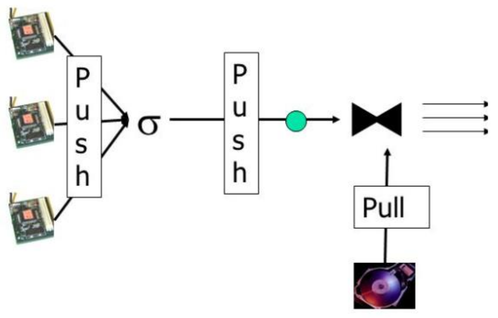

Figure 7: Fjord Query Processing Architecture

Ex: a) Pull from A to B: Suspend A, Schedule B until it produces data. A cannot go forward until B produces data. b)

Push from B to A: A polls, Scheduler thread invokes B until it produces data. A can process other input while waiting for B.

Operators:

Intially, non-blocking join operators can be utilized to perimit incremental joins over floods of data. Xjoin , , Eddy and Ripple Joins.

Xjoin Operator: XJoin Operator flushes records to disk if memory gets limited(during arriving phase).During a reactive phase, when data sources are blocked, XJoin utilizes already flushed tuples to deliver further join results. During the last cleanup stage after all inputs have been utilized, the XJoin operator joins the rest of the tuples that were missed during the previous stages.

***XFilter*:** It evaluates the collection of XML document which has similar user profile. In XFilter, the user interests are represented in XPath language The XFilter engine uses a efficient search index structure and a Finite State Machine (FSM) model to retrieve the user profiles. The frame work uses Selective Dissemination of Information (SDI) process for delivering the documents based on user requests. There are different basic of inputs in the framework are user **profiles** and **data items** (i.e., documents). User profiles deals the important information of individual users. In most systems these profiles are originated by the users through a Graphical User Interface. The user profiles are changed into a format that can be conveniently stored and assessed by the Filter Engine.

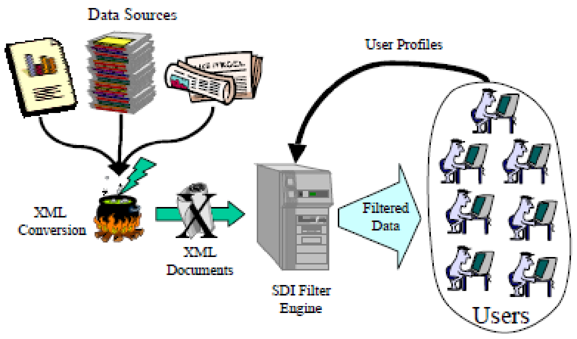

Figure 8: Xfilter Query processing system

The other key contributions to an SDI system are the records to be filtered. Our work is emphasis on XML-encoded documents.

The *Tukwila* system [IFF99] additionally underpins versatile adaptive query processing, in order to perform dynamic data integration over independent data sources. The Tukwila system supports operators that can be used to collect data over varying network conditions. The pipelined hash join function is used to integrate multiple hash table from disk. From the multiple point of view it resembles the hash ripple join like XJoin. A pipelined hash join works with two hash tables, as opposed to the single hash table of a usual hybrid hash join. The collector operator provides a robust strategy for integrating data from different sources with common schemas. This rule is indicated by an approach specified in Tukwila's rule language.

Eddy: Eddy is a query processing mechanism which continuously reorders operators in a query plan as it runs. The eddy architecture is quite simple, obviating the need for traditional cost and selectivity estimation, and simplifying the logic of plan enumeration

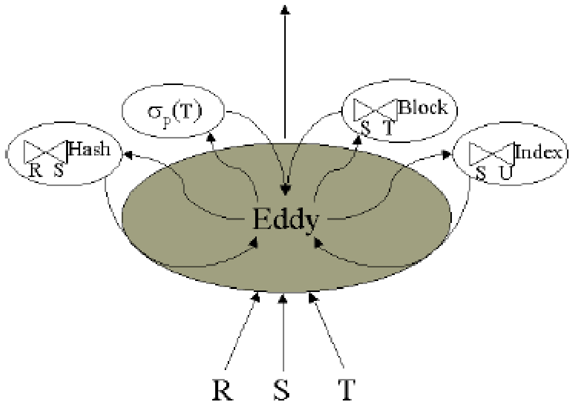

Figure 9: Eddies Query processing system

An eddy is a pipelined approach in which Data streams are collected from relations R,S and T. The eddy shows tuples to operators; the operators run as autonomous threads, returning tuples to the eddy. The eddy sends a tuple to the output only when it has been taken care by all the operators. The eddy adaptively selects an order to route each tuple through the operators.

B. NiagraCQ (Niagra Continous Query Processing)

NiagaraCQ query processing mechanism proposed continuous queries grouping techniques for efficient evaluation.NiagraCQ uses the following

Query processing mechanism to handle continuous query

1. This is a updatable and incremental group optimization approach with dynamic re-grouping. Continuous fresh queries are executed with the available query.
2. NiagarCQ uses a query-split scheme that instantiate negligiable changes to a broadly used query engine.
3. NiagaraCQ combines both change-based and timer-based queries in a consistent way.

The goal of the Niagara project is to develop a distributed database system for querying distributed XML data sets using a query language like XML-QL. Niagara monitors web XML sources and intermediate files on its local disk. It handles the disk I/O for both ordinary queries and continuous queries and holdup both **push-based and pull-based data sources**. For push-based data sources, the Data Manager is informed of a file change and notifies Event Detector actively.

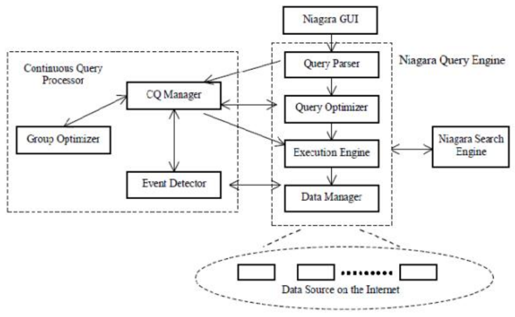

Figure 10: NiagaraCQ Query processing system

Architecture of Niagara CQ:

- A **continuous query manager**, which is the essential module of NiagaraCQ system. It facilitate a continuous query interface to users and initiates the Niagara query engine to execute fired queries. the Niagara data manager was enhanced to maintain the incremental evaluation of continuous queries

- A **group optimizer** that performs incremental group optimization.
- An **event detector** that detects timer events and changes of data sources.

C. Triggers

Triggers are *event-condition-action* rules, which are used to analyse the events detected in the transactions and perform the action accordingly. For launching trigger actions the procedures are performed by SQL data manipulation commands and user-defined stored procedures, continuous queries over *active tables*. Trigger processing is one of the prominent techniques in effective data management and processing techniques for continuous queries over data streams, such as specialized query optimization techniques.

D. Materialized Views

In the data stream environment, the materialized data view is used for managing data replica. In distributed environments, materialized views can be used to replicate data at distributed sites and maintains the updates regularly.

E. Sensor Proxy

The significant part of our sensor query is the sensor proxy, which acts as an interface between a single sensor and query system. The purpose of proxy is to protect the sensor from having to deliver data to several interested end-users. It acknowledges and services queries for the sake of the sensor.

One behavior of the sensor proxy is to modify the sample rate of the sensors, according to the user demand. In the event,If users are only interested in a few tests per second, there's no purpose behind sensors to sample at several hertz. An additional role of the proxy is to direct the sensor to aggregate samples in predefined ways.

Sensor proxies are ever ending services that exist across many user queries and map the tuples to different query operators based on sample rates and filter predicates specified by each query.

V. QUERY EVALUATION AND OPTIMIZATION

In the databases, a query optimizer is used to select the "best" query plan for query execution. A continuous query processor uses a execution plan for query optimization.

A. Pipelined operators

The selection operator can be used in data stream context. It is used to evaluate portions of the query multiple times, or use *Scratch* to hold temporary results during query processing

B. Blocking Operators

A ***blocking*** operator is one that processes all the rows completely before data is passed to other processes. Stream iterator is used to process punctuated data streams which requires initial state, step, pass, propagate, and purge functionality.

CONCLUSIONS AND RESEARCH PLAN

In this paper, we focused the various Continuous data stream processing techniques in WSN and we have an research idea in Continuous query processing which includes new Adaptive query processing, online aggregation and approximation techniques.

References

[1] Gupta and I. S. Mumick. Maintenance of materializedviews: Problems, techniques, and applications. *IEEEData Engineering Bulletin*, 18(2):3–18, June 1995.

[2] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *External Memory Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 50, 1999.

[3] H. Garcia-Molina,W. J. Labio, and J. Yang. Expiring data in a warehouse. In *Proc. of the 1998 Intl. Conf. on Very Large Data Bases*, pages 500–511, August 1998.

[4] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. Of the 2000 Annual Symp. on Foundations of Computer Science*, pages 359–366, November 2000.

[5] P. B. Gibbons, Y. Matias, and V. Poosala. Histogrambased approximation of set-valued query-answers. In *Proc. of the 1997 Intl. Conf. on Very LargeData Bases*, pages 466– 475, August 1997.

[6] Goetz Graefe. Encapsulation of parallelism in the volcano query processing system. In *Proc. of the 1990 ACM SIGMOD Intl. Conf. on Management of Data*, pages 102– 111, May 1990.

[7] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993. J. M. Hellerstein, M. J. Franklin, et al. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2):7–18, June 2000.

[8] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pages 261–272,May 2000.

[9] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. Technical report, June2001.

[10] J. M. Hellerstein, M. J. Franklin, et al. Adaptive query processing: Technology in evolution. IEEE Data EngineeringBulletin, 23(2):7–18, June 2000.