STATISTICAL DATA ANALYSIS OF CONTINUOUS STREAMS USING STREAM DSMS

Nadeem Akhtar

Department of Computer Engineering, Aligarh Muslim University, Aligarh, India
nadeemalakhtar@gmail.com

ABSTRACT

Several applications involve a transient stream of data which has to be modeled and analyzed continuously. Their continuous arrival in multiple, rapid, time-varying and possibly unpredictable and unbounded way make the analysis difficult and opens fundamentally new research problems. Examples of such data intensive applications include stock market, road traffic analysis, whether forecasting systems etc. In this study, we have used a Data Stream Management System tool- Stanford STREAM to model and analyze data from two different application domains- Road Traffic analysis and Habitat Monitoring analysis. Based on the results we discuss advantages and disadvantages of STREAM.

KEYWORDS

Continuous Stream, DSMS, Stanford STREAM

1. INTRODUCTION

The number of tools for analyzing continuous stream data is continuously increasing, because there is an increasing need in many different application domains. Several application domains require analysis of continuous data stream like stock market, security, telecommunications data management, web applications, manufacturing, sensor networks and others.

One example of such application is Road Traffic Analysis. Several countries have been using RFIDs to collect real time vehicular information [1]. RFID readers read information from RFID tags attached to vehicles and sent to a centralized server. For a large city, number of vehicles passed through a particular point may go into millions. Moreover, vehicular traffic is intrinsically unpredictable and time-varying. Another example is Network Packet Analysis. Network operators and service providers need to monitor data traffic to analyze the provided service level, to identify bottlenecks, and initiate appropriate counter measures, if possible [2][3]. This is especially important in the Internet, because the amount of data traffic continuously increases and the behavior and requirements of end-users are changing over time, like accepted response time from web servers.

Data stream management systems have been developed to monitor the continuously arriving data. They are different from traditional Database Management Systems in that they work on transient tables rather than persistent tables. Database Management Systems (DBMSs) may be used for analyzing continuous stream data. Traditional DBMSs, however suffer from some serious bottlenecks which limit their functionality from such real time complex applications requiring continuous monitoring of ever-changing data-streams. First, the DBMS is a passive repository storing a large collection of data elements and that humans initiate queries and transactions on

DOI: 10.5121/ijdms.2011.3206

this repository. This is called Human-Active, DBMS-Passive (HADP) model. Second, it is assumed that the current state of the data is the only thing that is important. Hence, current values of data elements are easy to obtain, while previous values can only be found torturously by decoding the DBMS log. The third assumption is that triggers and alerter are second-class citizens. These constructs have been added as an after-thought to current systems, and none have an implementation that scales to a large number of triggers. Fourth, DBMSs assume that data elements are synchronized and that queries have exact answers. In many stream-oriented applications, data arrives asynchronously and answers must be computed with incomplete information. Lastly, DBMSs assume that applications require no real-time services [4].

Data Stream Management Systems are specifically designed for handling continuous data streams. They can handle multiple, time-varying, unpredictable and unbounded streams which cannot be handled using traditional tools. In this paper, we have used a Data Stream Management System- Stanford STREAM in two different application domain namely Road Traffic analysis and Habitat Monitoring analysis. Section 2 presents the related work. The features of a general DSMS and Stanford STREAM are discussed in section 3 and 4 respectively. Section 5 describes the results obtained using Stanford STREAM DSMS to analyze the stream from abovementioned domains.

2. RELATED WORKS

Data stream management systems have been used in several application domains for mining continuous streams. Hebrail [5] provides a good survey of various data stream management systems and their applications. Following are the works which are similar to the work presented in this paper.

In [6][7], a TelegraphCQ prototype is studied and evaluated for network packet monitoring task. Some restrictions in the design of TelegraphCQ prototype are identified that makes it not suitable as a general tool for traffic analysis. Abdessalem [8] have compared STREAM and TelegraphCQ prototypes to analyze electric power consumption data. According to their experiments, it appears that TelegraphCQ is better adapted to the needs of their experiment than STREAM. In [9], a flexible network monitoring tool is described, called PaQueT, designed to meet a wide range of monitoring needs. PaQueT has been developed as an extension of Borealis Data Stream Management System and it can be easily extended to consider several types of network protocols. Chen [10] provide an overview on a DSMS prototype called T2 which inherits some of the concepts of an early prototype, Tribeca, but with complete new design and implementation in Java with an SQL-like query language. In [11], a data stream management system designed to be applied in medical monitoring systems is presented. In this paper, developed theory and query language is adapted to specific requirements of the signal processing in biophysical monitoring systems

3. DATA STREAM MANAGEMENT SYSTEMS

The DSMSs focus on querying streams of data instead of the database. As mentioned in the preceding sections, this causes some changes in the DSMSs' architecture compared to the previously mentioned list. Following, we compare the two systems' design to show the differences between them.

1. Instead of persistent relations, the DSMS aims to handle transient streams. This implies that disk storage is not an issue; the data enters and leaves the system at possibly high rates. Though, as this poses an architectural opposite to the DBMS, some DSMSs have integrated a DBMS as a part of it. This opens for the possibility of joining between streams and tables, for instance. Examples may be TelegraphCQ [12] and Stanford STREAM [13].

2. When the DBMSs access the data once for each query, the DSMS uses continuous queries, which are queries that continuously obtain tuples from the streams.

3. Sequential access. Since the data arrives as a stream, the DSMS reviews the tuples as a linear sequence, and does not have access to the data before or after the access interval. The DBMS data is stored in a database, and the data can be randomly accessed by specifying which blocks to read. In operations like aggregations and joins, this means that the DBMS blocks while performing these operations. This is not possible over a linear stream of data. When operating over streams, the DSMSs have to support windowing for blocking operators, which means that even though the stream is infinite, calculations are performed on small partitions of the stream.

4. Since the DSMS does not generally aim to store tuples as they arrive in the system, it is bound by the size of the available memory. This is also an issue with regard to disk I/O. As the streams may arrive at high rates, expensive and time consuming disk I/O can not be allowed. This means, as deducted in the description of the n-pass algorithms, that only one-pass algorithms can be used.

5. Due to optimizing, the DBMS has a set of rules that are introduced in the query rewriting process. An example of an optimizing rule is to push projections as far as possible down to the source, i.e., the database or the data stream. By doing this, less attributes are sent to the next operators, hence reducing the load. These rules also play a role in the DSMS, but the DSMS needs to adapt to the stream as well, by optimizing the query tree on the fly, or re-allocating queue sizes.

6. The DSMS is pledged to compute data and deliver results within a deadline, i.e., a time-limit. As data streams may arrive at high transfer rates, this may force the DSMS to throw away tuples that it can not compute, because of factors such as too complex queries. Generally, this means that not all the tuples may be computed, and that the DSMS has to support a set of approximation algorithms that deliver results that e.g. give a sample of the discarded tuples. DBMSs can not guarantee that the results - since they are required to be correct - will be displayed within a deadline. Complex queries using relations of several Gigabytes may take hours to terminate in a DBMS. The update rate also plays an important role in real-time processing. Compared to main memory processing, the disk-based update rate is limited due to the relatively slow hard disk. This means that the DSMS only gets the data from the network, performs computations on them, and then deletes them, to make room for new tuples. One or more streams enter the system and are processed by the query processor. State information might be sent between the input monitor and the query processor to inform about e.g. stream characteristics, such that optimizing and adaption may be performed. When the query operators are finished processing the tuples, the result is sent to the output buffer.

Since data stream management has been a hot topic the last few years, several systems have been developed. Some important ones are TelegraphCQ [12], Stanford STREAM [13], Aurora [14] and Niagra [15].

4. STANFORD STREAM

STREAM is developed at the Stanford University, and is a general purpose DSMS that aims to investigate resource sharing and adaptivity when for example several queries have common sub-expressions. STREAM supports declarative continuous queries over two types of inputs: streams and relations. A continuous query is simply a long-running query, which produces output in a continuous fashion as the input arrives. The queries are expressed in a language called CQL [16]. The input types-streams and relations are defined using some ordered time domain, which may or may not be related to wall-clock time.

Definition (Stream): A stream is a sequence of time stamped tuples. There could be more than one tuple with the same timestamp. The tuples of an input stream are required to arrive at the system in the order of increasing timestamps. A stream has an associated schema consisting of a set of named attributes, and all tuples of the stream conform to the schema.

Definition (Relation): A relation is time-varying bag of tuples. Here "time" refers to an instant in the time domain. Input relations are presented to the system as a sequence of timestamped updates which capture how the relation changes over time. An update is either a tuple insertion or a tuple deletion. The updates are required to arrive at the system in the order of increasing timestamps. Like streams, relations have a fixed schema to which all tuples conform. The output of a CQL query is a stream or relation depending on the query. The output is produced in a continuous fashion as described below:

• If the output is a stream, the tuples of the stream are produced in the order of increasing timestamps. The tuples with timestamp τ are produced once all the input stream tuples and relation updates with timestamps $\leq \tau$ have arrived.

• If the output is a relation, the relation is represented as a sequence of timestamped updates (just like the input relations). The updates are produced in the order of increasing timestamps, and updates with timestamp τ are produced once all input stream tuples and relation updates with timestamps $\leq \tau$ have arrived.

The STREAM architecture is made up of two broad components [17]:

1. Planning subsystem, which stores metadata and generates query plans, and
2. Execution engine, which executes the continuous queries.

STREAM currently does not support all the features of CQL. In this section, we mention the important features omitted in the current implementation of STREAM that we found based on our experiences with STREAM. The important omissions are:

1. Sub-queries are not allowed in the Where clause. For example the following query is not supported:
Select * From S Where S.A in (Select R.A From R)

2. The Having clause is not supported, but Group By clause is supported. For example, the following query is not supported:
Select A, SUM(B) From S Group By A Having MAX(B) > 50

3. Expressions in the Project clause involving aggregations are not supported. For example, the query:
Select A, (MAX(B) + MIN(B))/2 From S Group By A is not supported. However, non-aggregated attributes can participate in arbitrary arithmetic expressions in the project clause and the where clause. For example, the following query is supported: Select (A + B)/2 From S Where (A - B) * (A - B) > 25

4. Attributes can have one of four types: Integer, Float, Char(n), and Byte. Variable length strings (Varchar(n)) are not supported.

5. Windows with the slide parameter are not supported.

6. The binary operations Union and Except is supported, but Intersect is not.

5. STATISTICAL ANALYSIS OF CONTINUOUS STREAMS

We have used Stanford STREAM Data Stream Management System to monitor and analyze data streams from two application domains. In this section, we show and discuss results obtained.

5.1 Linear Road Benchmark Analysis

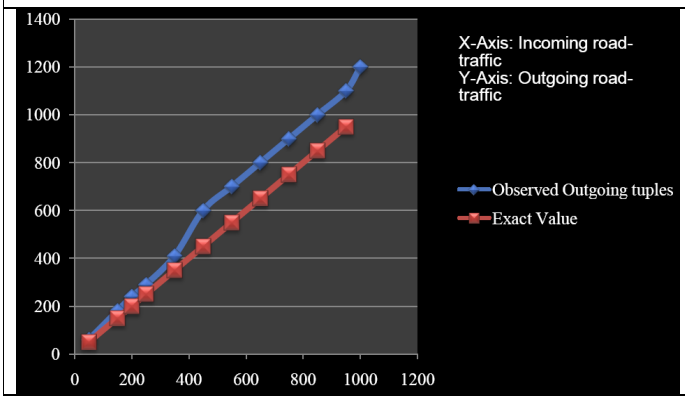
This section explains about the Linear Road Benchmark Analysis. We have taken a random data (approximated to real-life scenario) & performed a trivial road-network analysis & information retrieval on it. This domain is considered with the aim of providing useful information to the Road Traffic Department such as issues pertaining to fluidity of the road, average speed on segments of the road, tolling information & toll generation (based on traffic volume). The domain was successfully implemented and we were able to extract useful information from a single stream of incoming data. The input to the benchmark contains a "live" stream. We have considered nine expressways, each divided into eight horizontal segments as shown in Figure 1. The input stream is described below.

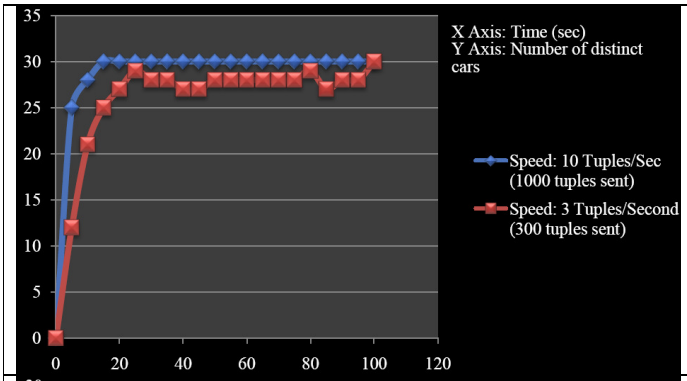
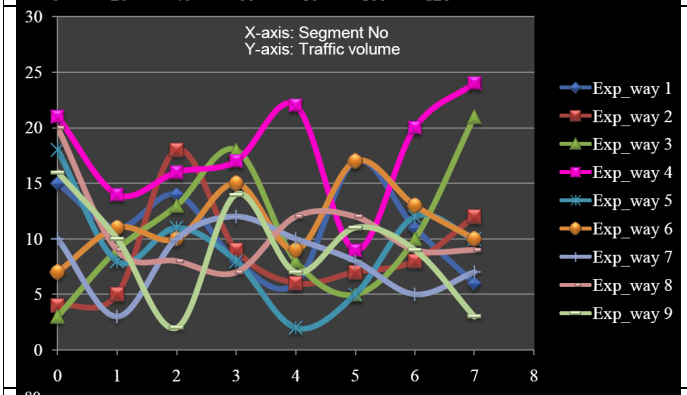
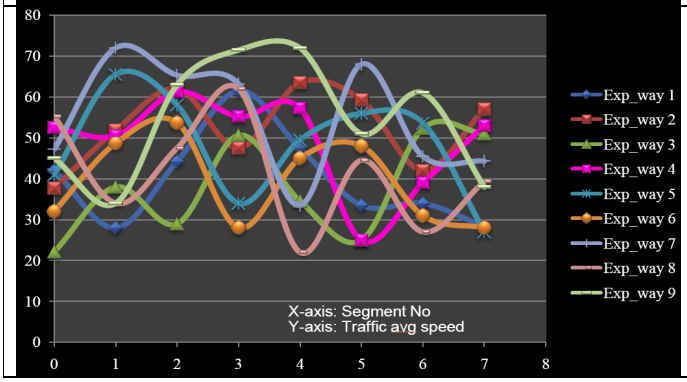
CarLocStr: Stream of car location reports. This forms primary input to the system.

```
CarLocStr( car_id, /* unique car identifier */
        speed, /* speed of the car */
        exp_way, /* expressway: 0..8 */
        lane, /* lane: 0,1,2,3 */
        dir, /* direction: 0(east), 1(west) */
        x-pos /* coordinate (segment) in express way */)
```

Table 1 summarizes output of queries executed and the purpose for which they can be used by the road traffic department. Figure 2 shows the STREAM's GUI interface executing a query. Since the data we have taken is not accurate so the results. But it serves our purpose to show the STREAM's ability to be used in this domain.

Table 1. Results and Conclusion of queries executed over CarLocStr Stream

Figure	Information Retrieved	Conclusion Derived
	Incoming and Outgoing Road Traffic	Help determine the number of vehicles on road

	Number of distinct cars simultaneously active during the range window of 30 seconds	Determines the number of different vehicles on road & helps in monitoring the total traffic
	Number of cars in different segments of each expressway	Helps in GPS applications like Tomtom [18] that helps in determining the best possible route for travelling
	Average speed in different segments	Helps in GPS application like Jamdroid [19] that equate speed with the traffic on segment

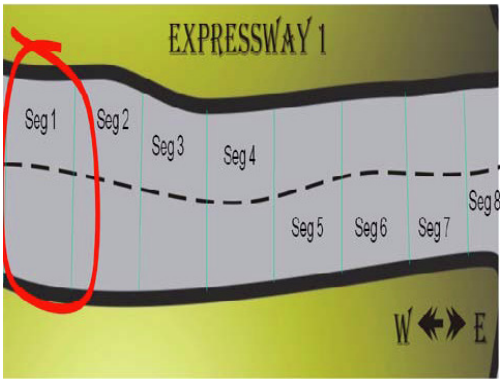


Figure 1. Each Expressway Divided into Eight Segments

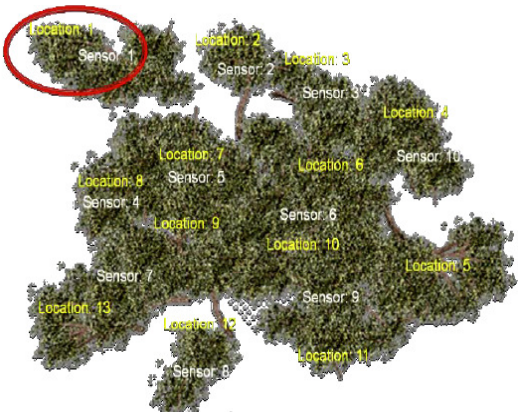


Figure 2. Hypothytical Layout

5.2 Habitat Monitoring Analysis

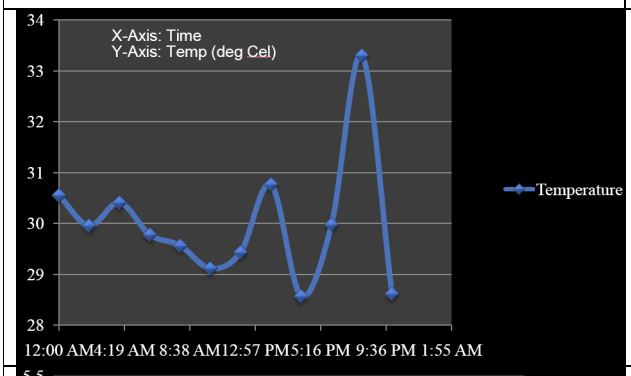
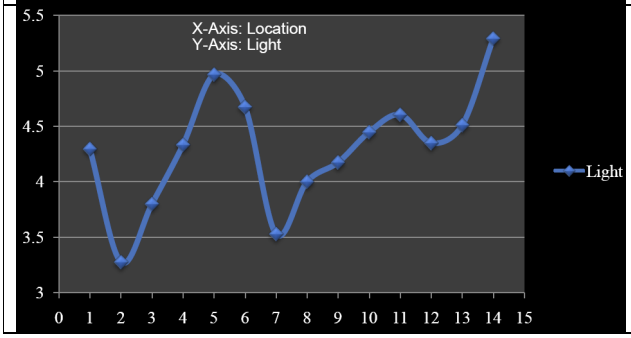
Researchers are deploying sensors in and around the nesting holes of birds on Great Duck Island in order to collect detailed data for studying the nesting habits of these birds. The temperature, humidity, barometric pressure, and infrared sensors that have been deployed relay readings to a computer base station. The base station feeds the stream into a satellite link enabling researchers to access the data in real-time over the Internet [20]. We have considered a random data set of a forest area approximated to real life scenario. Entire forest area is divided into 15 locations, where one sensor is installed per location. Figure 3 shows the hypothetical layout of the forest area. For simplicity, we have assumed the sensor data to consist of only light intensity and temperature recordings.

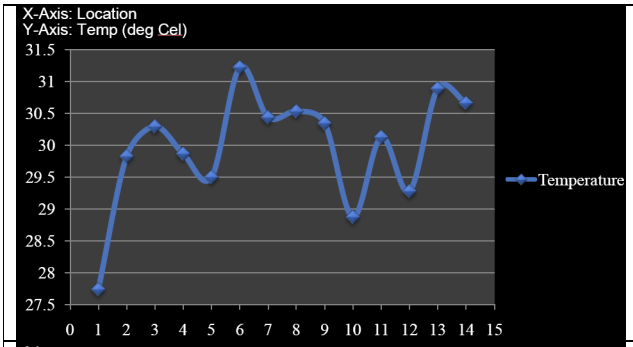
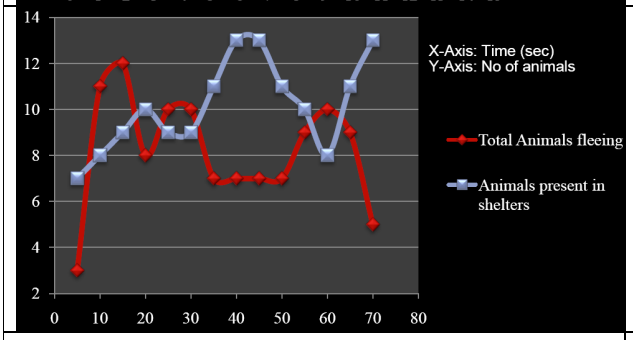
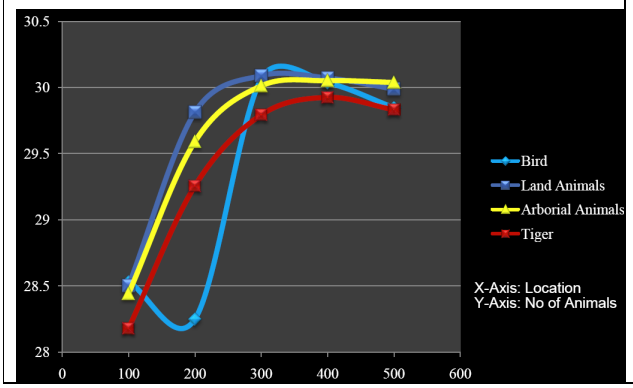
Sensors (Stream): Sensor motes measure light, temperature and produce a data stream with the following schema.

```
Sensors ( id /* unique identifier of the sensor */,
          location /* the location of the sensor */,
          light /* the current light reading */,
          temperature /* the current temperature reading */,
          timestamp /* time of measurement */);
```

Table 2 summarizes output of queries executed over the Sensors Stream and the purpose for which they can be used by the forest department.

Table 2. Results and Conclusion of queries executed over Sensors Stream

Figure	Information retrieved	Conclusion derived
	Average temperature of the habitat monitored over 24 hrs	Helps us in inferring whether the area is suitable for particular species
	Average light intensity for different locations	Helps us in inferring whether the area is suitable for particular species

	Average temperature for different locations	Helps us in finding the most suited of all the location for a particular species
	Flow (in & out) of animals in the intervals of 10 second	Monitoring the habitat area in terms of total animals coming in & going out
	Average temperature at which animals are found	Useful for biologist researchers in determining the ideal breeding & living temperatures for different animals

6. CONCLUSION AND FUTURE WORK

The Stanford STRAM DSMS, which is appropriate for an envirnment with vast amount of continouous stream data, is discussed and used in Road Traffic analysis and Habitat Monitoring analysis. Based on our experiences with STREAM, we can easily say that it easily covered and exceeded our expectations. Below is the list of advantages that we think make STREAM a suitable applications in real life streaming applications:

• Not a single tuple was dropped (Tested till 20000 t/s): The fact that STREAM displays such levels of robustness, easily makes it one of the best DSMS tools around. It also makes STREAM highly suitable for extreme precision applications like Stock exchange streams, weather forecasts, etc.

• Extremely accurate on aggregation operations: The error percentage in our working environment varied from 0.125% to 0.025% thereby again portraying STREAM as an accurate DSMS tool giving reliable output.

• Supports a sub-set of SQL queries that are easy to understand: As against ad-hoc development & deployment of conventional stream handling tools, STREAM offers its user CQL which is easy to use with users with previous SQL experience.

• GUI environment: Only DSMS tool identified by us which had GUI environment. This makes STREAM user friendly & coupled with the fact that it is easier to install & deploy, certainly makes it one of the better DSMS around.

• Generates query plans: This makes STREAM more user interactive & shows graphically the relational model of the project.

• Based on client-server architecture: Many clients can simultaneously access the server resources.

In spite of being quite useful for continuous stream management tasks, STREAM shows some loopholes during our experiments. These are listed below:

• Robustness level drops severely when number of simultaneous queries increases to about 8 & above: The system hangs frequently at registries with relations that are strongly dependent on each other

• System crashes frequently on some aggregation operations like min, max. The support for these aggregation operators is extremely limited.

• Requires conversion of data stream to text file before operation could be performed: Instant operation on live streams is still not supported & hence real-time analysis could not be performed on streaming data.

• System crashes on complex queries at high speed: System robustness drops severely at high speed coupled with relations that are complex & related

• Tuple duplicity because of tuple redundancy: STREAM inputs the results at first and then exits it at next interval. This fact introduces tuple redundancy as tuple accumulation occurs at time when the tuple is not meant to be in the system.

• Supports only a small subset of SQL queries as discussed before.

Following improvements could be made in the STREAM DSMS:
1. STREAM could be made real time by taking streams as input rather than text files. Inputs should be enabled to be taken from sensors
2. STREAM should support a wider range of SQL queries
3. Robustness levels should be increased & redundancy should be minimized. Tumbling window support should be enabled.
4. Relations should be allowed to be formed at real time.

REFERENCES

[1] Chattaraj, Saumya Bansal, and AnirudhhaChandra. An intelligent traffic control system using rfid. IEEE Potentials, 28(3):40–43, 2009

[2] Mohammed Qadeer, Nadeem Akhtar, Faraz Khan and Farid Haque, “Monitoring and Analysis of Data Packets using Data Stream Management System”, ICCEE 2008, 20 - 22, December 2008, Phuket Thailand

[3] Thomas Plagemann, Vera Goebel, Andrea Bergamini, Giacomo Tolu, Guillaume Urvoy-Keller and Ernst W. Biersack, "Using Data Stream Management Systems for Traffic Analysis – A Case Study", Lecture Notes in Computer Science, 2004, Volume 3015/2004, 215-226

[4] Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S.,Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.,:Monitiring Streams- A New Class of Data Management Applications

[5] G Hebrail, "Data Stream Management and Mining", Book Chapter, Mining massive data sets for security: advances in data mining, search, Social Networks and Text Mining, and their Applications to Security, Volume 19, 2008, ISBN 978-1-60750-362-0

[6] T Plagemann, V Goebel, A Bergamini, G Tolu, G Urvoy-Keller, E W. Biersack, "Using Data Stream Management Systems for Traffic Analysis- A Case Study –" Passive and Active Network Measurement , Lecture Notes in Computer Science, 2004, Volume 3015/2004, 215-226, DOI: 10.1007/978-3-540-24668-8_22

[7] Nadeem Akhtar, Mohammed A Qadeer, Faraz Khan, Faridul Haque, "Data Stream Management System: Tools for Live Stream Handling & their application on trivial Network Analysis Problems", International Conference on Innovations in Information Technology, 2008. IIT '08, 16-18 Dec. 2008.

[8] Talel Abdessalem, Raja Chiky, Georges H´ebrail, Jean Louis Vitti, "Using Data Stream Management Systems to analyze Electric Power Consumption Data", Workshop on Data Stream Analysis – San Leucio, Italy - March, 15-16, 2007

[9] Ligocki, N.P.; Hara, C.S.; Lyra, C., "A flexible network monitoring tool based on a data stream management system", IEEE symposium on computers and communications, 6-9 July 2008 (ISCC 2008)

[10] Chung-Min Chen, Hira Agrawal, Munir Cochinwala, David Rosenbluth, "Stream Query Processing for Healthcare Bio-sensor Applications," icde, pp.791, 20th International Conference on Data Engineering (ICDE'04), 2004

[11] Widera, M.; Wrobel, J.; Owczarek, A.; Matonia, A.; Jezewski, M.; "Data Management System for Computer Aided Biophysical Monitoring ", 27th Annual International Conference of the IEEE, Engineering in Medicine and Biology Society (EMBC), 2005

[12] TelegraphCQ: http://telegraph.cs.berkeley.edu/, 2008

[13] Stanford STREAM website: http://wwwdb.stanford.edu/stream

[14] www.cs.brown.edu/research/aurora/

[15] http://datalab.cs.pdx.edu/niagara/

[16] Arasu, S. Babu and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution, *VLDB Journal, 2005*

[17] STREAM: The Stanford Stream Data Manager -User Guide and Design Document, www.infolab.stanford.edu/stream/code/user.pdf

[18] www.tomtom.com

[19] Mohammed A Qadeer, Nadeem Akhtar, Faraz Khan, Etienne Baratte, "Improving Real-Time GPS by incorporating TelegraphCQ in Jamdroid Architecture", ISWPC 2009, 11-13 Feb 2009, Melbourne, Australia

[20] http://infolab.stanford.edu/stream/sqr/birdmon.html#queryspecseng

Author

Nadeem Akhtar received his B. Tech and M.Tech degree in Computer Science from Aligarh Muslim University, Aligarh, India. He is currently working as Assistant Professor in the department of Computer Engineering, Aligarh Muslim University. His research interests include Data Mining, Database Management System and Operating Systems.