Data Stream Management System and Capital Market Surveillance

Aniruddha Mukherjee, Prasun Bhattacharjee, Debnath Mukherjee, and Prateep Misra

***Abstract*—Importance of real-time data analysis has been felt since early '90s and thus processing of streaming data (from either sensor networks or telecom switches or web and other disparate systems) is the demand of the industries worldwide. Quicker detection of fraudulent activities in a financial system is the order of the day. Thus capital market surveillance, if can be performed by using the streaming input of various trading transactions, without being stored, that would be beneficial to the regulatory authorities and stock exchanges. In this paper, we describe how stream processing using a data stream management system (DSMS) can be used for the above task and how effective would be that in terms of performance and latency. We present results obtained from using a commercial event stream processing system (IBM InfoSphere Streams platform) for certain typical fraud detection scenarios.**

***Index Terms*—Capital market surveillance, data stream management systems, high performance. low latency, stream processing.**

I. Introduction

In a capital market, investors and all sorts of market-participants expect a transparent, influence-free, open market for trading stocks, options, commodities etc. The responsibility to ensure the above lies on the exchanges where instruments are traded. They are guided by the regulatory authorities and as per the laws of the land. But history of market manipulations seems to be a phenomenon since the inception of these kinds of markets. Thus capital market surveillance is a highest priority activity for the exchanges and regulatory authorities.

In modern world, all financial transactions happen electronically through web-based systems. This has enabled many people to perform trading in the capital market. As a result, huge amount of data gets generated from various sources and the rate of such generation of such data-streams is very high. Therefore, there is a pressing demand to detect market manipulations by identifying anomaly in trading as fast as possible i.e. in a near-real time mode. On the other hand, while the techniques or algorithms for such detection are though available to the authorities but constraints for executing them on stream of real-time data are posing obstacles. In this paper, we will describe in brief what a data stream management system (DSMS) is and a novel way to use a DSMS platform for some methods of capital market surveillance.

Manuscript received March 6, 2012; revised May 5, 2012.

The authors are with the Innovation Labs, Tata Consultancy Services Ltd Bengal Intelligent Park Ltd. Bldg # D, Salt Lake Electronic Complex, Kolkata, India (e-mail: aniruddha.mukherjee@tcs.com, prasun.bhattacharjee@tcs.com, debnath.mukherjee@tcs.com, prateep.misra@tcs.com).

This paper is organized as follows. In Section 2 we introduce the definitions of Event Processing and Stream Processing. In Section 3 we narrated briefly various projects which dealt with stream management or data stream processing. In Section 4 we discuss approach towards real time capital market surveillance. In Section 5 we narrated the scenarios for real time capital market surveillance.

Finally in Section 6 we present results of our study of use of stream processing in capital market surveillance in a lab environment. Six scenarios of capital market transactions have been considered. In each of these scenarios our Stream Processing system processes trade data in real time and tries to detect aberrations in trading pattern. We run the system with sufficiently large volume of real data form a major stock exchange. Our fraud detection solution has been implemented through IBM InfoSphere Streams product. We compare the results obtained from this solution with that obtained from another one that uses an open source Java based Complex Event Processing (CEP) product called ESPER. The result obtained from using the Streams is shown to be superior.

II. Stream ProcessinG

A. Event Streams, Windows and Event Stream Processing

According to the Event Processing Glossary [1] published by the Event Processing Technical Society, any phenomenon happening or contemplated as happening can be treated as an event.

An event stream is a linearly ordered sequence of events and usually ordered by time [1] i.e. it flows as vectors {data-tuple, timestamp} and a component of the structure of an event is called an event attribute.

A bounded portion of an event stream is called a window [1] of event and thus a window defines a subsequence of an event stream.

The event stream processing is defined as the computing on inputs that are event streams [1]

Companies can improve the timeliness, agility, and information quality of their operations if they handle events in a systematic way that leverages advances in the contemporary understanding of how events workFinal Stage

When you submit your final version, after your paper has been accepted, prepare it in two-column format, including figures and tables.

B. Data Stream Management Systems

Data management scientists have understood the importance of a new class of data-intensive applications that requires managing data streams, i.e., data composed of

continuous, real-time sequence of items. Streaming applications pose new and interesting challenges for data management systems. Such application domains require queries to be evaluated continuously as opposed to the one time evaluation of a query for traditional applications. Streaming data flows/arrives continuously and queries must be evaluated on such unbounded data sets. Data Stream Management System or DSMS (also called SDMS Stream Data Management System), is a system to deal with high volume of unbound data streams where the data is provided on real-time and continuous basis. The platform which enables processing of stream-data is called Stream Processing Engines or SPE.

SPEs use specialized primitives and constructs (e.g., time-windows) to express stream-oriented processing logic and supports SQL operations on streams (as well as on stored data simultaneously). SPEs offer the best capabilities since they are designed and optimized from scratch to address the requirements of stream-data processing. On the other hand, paradigms of DBMS and Rule Engines were originally architected for a different class of applications with different underlying assumptions and requirements [2].

C. Architecture of a Data Stream Management System

Fig. 1 shows the architectural layers in a DSMS. A DSMS has the following features or properties:

- Query is registered and persistent as continuous query
- Does sequential access to data (in streams); hence data is transient
- Resource (mainly memory) is limited and query evaluation is done through one-pass method – continuous update of results
- Query plan is adaptive and hence query answer is sometimes approximate; whereas processing speed is critical to produce in near real-time results

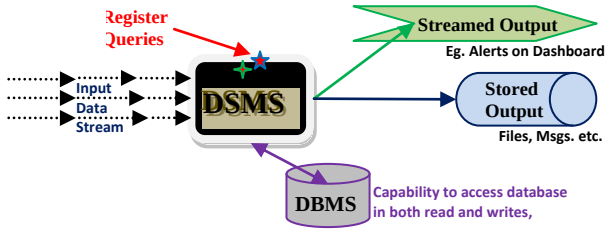

Fig. 1. Architecture of a data stream management system (DSMS)

III. Various Stream Processing Projects

A number of requirements must be met by any given system in order for it to be considered as a stream processing engine. These requirements include high availability, scalability, and optimizations. As well as the desired capabilities which includes real-time computation on the data contained within an event, and the ability to detect and possibly react to simple and complex events instantly is also important [3].

Huge amount of efforts were devoted all over the globe since late 1990s to create an efficient stream processing platform and define a language which would be an extension of Structured Query Languages (SQL) and capable of handling data in streams, in continuous manner [4]. The main areas where it was implemented are telecom call record applications, Sensor network, Network security, financial applications and Web-usage log application. A few of them are narrated briefly in the following paragraphs.

Tribeca (1997): A project at Bellcore, USA for network traffic analysis. It is a software system for querying arbitrarily long streams of information from a live network feed or from a tape or from a disk (i.e from a single source) and applies compiled queries to the stream

XFilter (2000): Content based filtering system for XML documents. It is a high-performance, scalable selective dissemination of information (SDI) system and uses a language called XPath.

NiagraCQ (2000): A project at University of Wisconsin-Madison and its goal is to is to develop a distributed database system for querying distributed XML data sets using a query language like XML-QL.

Xyleme (2001): Its goal was to build a dynamic warehouse for massive volume of XML data obtained from web. It uses content based filtering system for high throughput.

Hancock (2001): A language developed at ATandT Labs, for processing large-scale data. It is C based and designed to describe signatures of callers. It provides data abstraction mechanism and can control abstraction to facilitate looping over records. This can be used for various mass surveillance applications too.

Cougar (2002): Object based querying paradigm and used a declarative query language with object oriented stream modeling. This project was supported by DARPA and Cornell Information Assurance Institute.

TelegraphCQ (2002): It is a project of Berkley Database Research group of University of California and uses a relational based querying model which is called Continuously Adaptive Continuous Queries (CACQ).

STREAM (STanfordstREamdatAManager, 2003): It belongs to relational based querying paradigm and was a prototype DSMS developed by Stanford University, which used CQL (continuous query language).

Aurora (2003): It's a collaboration of Brandies University, Brown University and MIT for a scalable distributed stream processing. It uses procedural model of querying where users can specify query plan and data flow.

The Linear Road Benchmark (2004): designed by Aurora team and STREAM team – a typical simulation prototype for DSMS.

Borealis (2005): It superseded the Aurora project and is distributed multi-processor version of Aurora. A giant network of operators, each node has query processors (QP), high-availability is possible and here dynamic revision of query result is possible.

OpenCQ (2007): A project of Gerogia Tech University which has produced a distributed data stream processing system. It organizes the nodes into virtual network hierarchy.

Tapestry (2002): Xerox Palo Alto Research Center had built this system, which was designed to support *collaborative filtering*. This means that people collaborate to help one another perform filtering by recording their reactions to documents they read. Such reactions may be that a document was particularly interesting (or particularly uninteresting). These reactions, more generally called *annotations*, can be accessed by others' filters.

Infosphere Streams (2010): It is an IBM's commercial product. It provides an execution platform and services for user-developed applications that ingest, filter, analyze, and correlate potentially massive volumes of continuous data streams. It supports high volume, structured and unstructured streaming data sources such as images, audio, voice, VoIP, video, TV, financial news, radio, police scanners, web traffic, email, chat, GPS data, financial transaction data, satellite data, sensors, badge swipes, etc. However, it should be installed on a Linux O/s. All sorts of operations on streams are done through a language called stream processing language (SPL).

IV. Real Time Capital Market Surveillance Using Stream Processing Engine (SPE)

The rate at which the capital market transactions are generated is extremely fast – often going up to more than a million messages / second. Traditional approaches of storing the data in a data warehouse and then processing it via statistical analysis and data mining packages are often inadequate at these data rates. However, the problem of surveillance or fraud detection on streaming market feeds can be addressed through the efficient use of an SPE.

How IBM's InfoSphere Streams has been used in various large scale high-performance low-latency stock market applications for the market makers, have been given in [5] and [3].

An example of how monitoring of trading activity in an exchange is done on streaming data is shown in Fig. 2 (below). The system consists of a stream processing engine that takes as inputs all orders that are being placed by market participants. All orders pass through two filters, one which tracks all executed (or retained) orders and the other tracks all cancel orders. The outputs of these filters are two events-streams, namely the "Executed Order Stream" and the "Cancel Order Stream". These two streams are then input to the correlation engine that matches executed (or retained) and cancel orders from the same participant and accumulate the matched values for specified time windows (say X days or Y hours). Whenever the ratio of total cancel to total executed (or retained) orders exceeds a particular threshold, an alert is output. The generated alerts can then be further investigated by the surveillance department of the exchange.

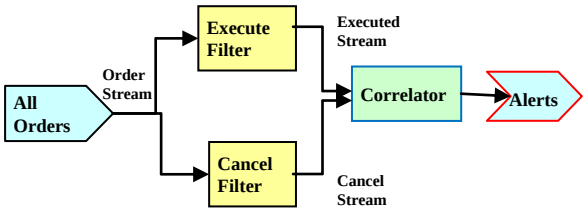

Fig. 2. Monitoring orders using an SPE

V. Market Surveillance PoC

We now present the details of a proof of concept project that investigates the effectiveness of SPEs in real time surveillance applications [6].

The following six scenarios as mentioned below were tested. Trade data from a major stock exchange was used to test the scenarios:

A. Long gap with Last Traded Date

An alert is generated when the trading for a security happens after a gap of "d" days, where "d" is a predefined threshold.

B. Anomaly in Average Trade Price and Quantity

An alert is generated when today's average trade price of a security deviates more from yesterday's average price by x% and its today's trade volume deviates more from yesterday's trade volume by y%. Here "x" and "y" are predefined thresholds.

C. Anomaly with Respect to Normal Values

An alert is generated when the abnormal trading activity for price, volume and trade count happens in a security in the following manner

Trade Count > Normal Trade count on a day
AND
[Trade Volume > Normal Volume for the day
OR
Trade Price > Normal Price for the day]

The normal values for a particular symbol can itself be learnt by the system based on past statistics. Normal values are those values that do not exceed statistical deviation bounds.

D. High-Low Variation

It is defined as the variation between the high price (H) and the low price (L) of a security, during a trading session, expressed as a percentage of the previous close price (P).

i.e. High-Low Variation = $\{(H - L)/P\} \times 100$.

An alert is generated when the high-low variation exceeds a predefined threshold.

E. Price Variation

It is defined as the variation between the last trade price (LTPt) and the previous close price (P) of a security expressed as a percentage of the previous close price (P).

i.e. Price Variation = $\{(LTPt - P)/P\} \times 100$.

An alert is generated when the price variation exceeds a predefined threshold value.

F. Consecutive Trade Price Variation

It is defined as the variation between the last trade price (LTPt) and the previous trade price (LTP_{t-1}) of a security expressed as a percentage of the previous trade price (LTP_{t-1})

i. e. Consecutive Trade Price Variation $\Delta LTP = \{(LTPt - LTP_{t-1})/LTP_{t-1}\} \times 100$.

An alert is generated when the ΔLTP exceeds a predefined value threshold.

The scenarios 4, 5 and 6 above have been detailed in [7].

For each of the scenario, a file containing real life trade data was used as input. The data in the file was read by a special operator that streamed the data to the processing system – thus simulating a real-time market feed. Another file that contains the summary data of the immediately previous day served as the second input to the system. This file provided static/fixed inputs required for the stream processing system. The third input was a file containing the list of selected symbols. Output alerts were saved in an

output file.

Diagrammatically, the process flow of a typical scenario of ours is shown in Fig. 3, below:

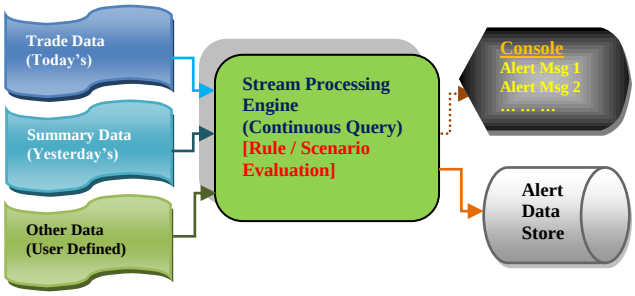

Fig. 3. Experiment process flow

VI. Test Setup and Results

The test setup and results obtained are described in the following sections.

A. Hardware and Software Used

Hardware – An Intel Xeon E5504 based server having 2 CPUs, each with four cores (total 8 cores) and RAM of 12 GB and six SAS disks each 146 GB capacity

Software – The operating system was Red-Hat Enterprise Linux (ver. 5.4); the programming languages used were SPL (Stream Processing Language), C++ and SQL; as the stream processing engine, we used IBM "Infosphere Streams" (ver. 1.2), Esper (an open-source product) which is an Java-based Complex Event Processing (CEP) engine and used for comparison purposes with SPE.

B. Results Obtained

The results of running the scenarios described in Section 7 are listed in Table I. It also depicts the time taken to process 2.382629 million input records for each scenario through the Infosphere Streams and also the time to process a record/tuple (in microsecond) are shown there.

TABLE III: Performance Results under Infosphere Streams

Scenario	Records read and processed	Time taken to complete the run	Time taken to process 1 tuple (µsec)
1. Long Gap	2,382,629	10 sec 695 millisecond	4.5
2. Price and Quantity anomaly	2,382,629	19 sec 31 millisecond	8.0
3. Abnormal Trades	2,382,629	18 sec 569 millisecond	7.8
4. High-Low Variation	2,382,629	5 sec 449 millisecond	2.3
5. Price Variation	2,382,629	18 sec 63 millisecond	7.6
6. Consecutive Trade Price Variation	2,382,629	5 sec 810 millisecond	2.4

We compared the results obtained with an SPE based solution, with a CEP based solution. The same scenarios were tested with a solution based on the Esper CEP. The following Table II shows the performances of the above six scenarios under Esper CEP (columns are same as in Table I).

TABLE II: Performance under Esper CEP Platform

Scenario	Records read and processed	Time taken to complete the run	Time taken to process 1 tuple (µsec)
1. Long Gap	2,382,629	41 sec 603 millisecond	17.461
2. Price and Quantity anomaly	2,382,629	40 sec 973 millisecond	17.197
3. Abnormal Trades	2,382,629	57 sec 723 millisecond	24.227
4. High-Low Variation	2,382,629	25 sec 843 millisecond	10.846
5. Price Variation	2,382,629	36 sec 939 millisecond	15.503
6. Consecutive Trade Price Variation	2,382,629	29 sec 187 millisecond	12.250

Clearly the performance under Esper CEP is 2.2 to 4.7 times inferior to that under Infosphere Streams SPE. While Infosphere Streams can process a tuple/record within 2.3 to 8 microseconds, the same takes 10.8 to 24.2 microseconds under Esper CEP's DSMS. This also establishes the fact that Java is about 3 times slower than 'C'.

We have further experimented to know the processing time taken for each and every tuple under the Streams platform and after analyzing the outcome we obtained the following figures as given in the Table III below.

TABLE III: Different Statistics for Time Taken To Create Output Tuples

Scenario	Different Statistics of time taken to generate output tuples (in microsecond)							
	Min.	Max.	Mean	Median	95th percentile	98th percentile	99th % percentile	99.9th percentile
1. Long Gap	3	106	4.33	4	5	6	6	8
2. Price and Quantity anomaly	6	204	7.23	7	9	9	10	13
3. Abnormal Trades	4	921	5.44	5	7	7	8	11
4. High-Low Variation	4	89	5.95	6	8	8	8	12
5. Price Variation	1	14	2.17	2	3	3	3	4
6. Consecutive Trade Price Variation	3	40	5.11	5	6	6	7	8

From the above statistics, we conclude that 99% of the stream-tuples were processed within 10 microseconds. The figures in the "Max." column are such larger than the "Mean" or "Median" because of delays in inter process communications, which was caused by the various daemon process running under the Linux O/s.

VII. Future Work on Stream Processing in Financial Domain

Primarily due to a lack of clear-cut criteria for identifying anomalous activity in financial domain, various theoretical methodologies after being applied in this domain, mixed results are disseminated.

We propose to continue research work on applying some of the data mining techniques, as described in [8] for data in streams to acquire knowledge and insights about such financial data stream. These stream mining can help in defining effective fraud detection criteria. Few points towards this direction are briefly given below [6]:

- Statistical distribution based mining through which mean. standard deviation of stock-price or VWAP can be estimated and co-related in real-time with historical data and/or current prior windows of data
- A pattern finding exercise on order placements/cancellations may lead to a strong association rule for the surveillance upon a specific set of market participation
- Real-Time Streams Mining would provide us knowledge in formation of clusters on some attribute(s) in stock trading like buy/sell orders placed, buy/sell orders matured to trades, cancelled buy/sell orders etc.
- Depending on the above clustering, one may define conditions of outlier detection

VIII. Conclusion

Deployment of stream processing engines can be beneficial to financial transaction processing systems since it delivers high performance and low latency. Therefore, its benefits may be reaped in market surveillance applications as well, where it can provide attractive performance for the industry people. By suitable arrangements of computing nodes and cores within those nodes, performance can further be enhanced by reducing latency to a great extent.

Acknowledgment

We would like to thank IBM for providing the InfoSphere Streams product to use at our labs and Mr. Senthil Nathan, Research Member of IBM, T J Watson Labs., USA for his continual support while developing the solution. We would also like to thank Mahesh Jambunathan, T. Arul and members of the TCS BaNCS Market Infrastructure Group, for their valuable inputs and review of the solution.

References

[1] Glossary of Terms (ver. 1.1). (July 2008). D. Luckham and R. Schulte.[Online]. Available: http://www.epts.com (last accessed on 24-Sep-2010).

[2] M. Stonebraker U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, issue 4, pp. 42-47, December 2005.

[3] X. J. Zhang, H. Andrade, and B. Gedik et al., "Implementing a high-volume, low-latency market data processing system on commodity hardware using IBM middleware," in *Proc. of 2nd Workshop on High Performance Computational Finance*, Article No. 7, Portland, Oregon, USA, 2009.

[4] M. Dylan, "An Analysis of Stream processing Languages," Department of Computing, Macquarie University, Sydney – Australia, 2009.

[5] B. Gedik, H. Andrade, K. L. Wu, P. S. Yu, and M. C. Doo, "SPADE: The System S declarative stream processing engine," in *Proc. of SIGMOD'08*, pp. 1123-1134, June 2008, Vancouver, BC, Canada.

[6] A. Mukherjee, P. Diwan, P. Bhattacharya, D. Mukherjee, and P. Misra, "Capital Market Surveillance using Complex Event Processing Technology," in *Proc. of 2nd ICCTD 2010 Conference*, November 2010, Cairo, Egypt, pp. 577-582.

[7] Surveillance in Stock Exchanges Module Workbook. [Online]. Available: http://www.nseindia.com.

[8] J. Han and M. Kamber, "Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers," *2nd Edition*, 2008.

Aniruddha Mukherjee is currently an employee of Tata Consultancy Services Limited, India, which is largest software exporter of Asia. This author is an masters degree holder of Statistics (M. Stat.) from Indian Statistical Institute, Kolkata, India in 1984 and has about 24 years of experience in information technology industry. Apart from leading software projects of various business domains and hardware platforms, he has been involved with innovation labs. of the company, where real-time analytics related work are being experimented for high-performance, low-latency aspects.