SQL & Stream Processing

Ashvita Hadge (W1270748), Chetan Pangam (W1282207),
Sai Lalitha Sree V (W1270081), Snehal Patil (W1159432)
Department of Computer Engineering.
Santa Clara University
Santa Clara, USA

Abstract—Many internet based applications produce huge data streams. We cannot use traditional data processing techniques because of huge real time data processing. Stream data is dynamic which is being continuously updated from environment. The traditional data processing cannot handle complex and numerous queries on a data stream. In this paper, we will discuss effective and efficient way to handle data stream processing. Another aspect is that the initial stream applications had quite simple requirements in terms of query processing. This made the existing DBMS systems look overloaded with functionalities. These factors led researchers to design and build new architectures from scratch and several Data Stream Management System solutions have been proposed over the last years.

Keywords— Stream Processing; Apache Spark; Apache Kafka; Big Data, Aurora, Data Stream .

I. Introduction

The ever-growing data in real time has increased pretty drastically. Real time data or market data generates tens of thousands of messages per second. Furthermore the data in electronic trading cannot accept even a single second of latency. Results has to be accurate or it will sabotage the transaction. This fact is causing financial services companies to require high volume of data processing with very low latency and this can be only be done using stream processing systems. Data in stream processing is continuous and in record by record fashion. Stream processing treats data as infinite stream of data integrated both from past and the current sources. It analyzes the continuous queries and then act on real time streaming data.

Ability of continuously calculating the mathematical or statistical analytics on the fly within the stream is essential for stream analytics. As it can handle high volume of data and is built using fault tolerant architecture, it enhances and enables the analysis of data which is continuous.

The traditional database first stores the data and then runs queries over it for analysis but since stream data is inbound data, it has to queried on the fly as it streams through the server. Stream processing also connects to external data sources, enabling applications to incorporate selected data into application flow, or to update an external database with processed information.

II. Motivation

Stream processing is one of the emerging fields. Applications requiring real-time processing of huge data sets are pushing limits of the traditional data processing infrastructures. The examples of these applications which require real time streaming are network traffic analysis, trading done at wall street, fraud detection, sensor networks, data sent over for self-driven cars, command and control in military environments. Querying such real time data and also storing them for data mining purposes is highly computationally challenging. Stream applications has lead to the birth to the specialized stream engines which has gained significant popularity over the past years. These systems are developed on an entirely new concepts as compared to the traditional database engines to cope up with the ever emerging data streams that arrive at the moment. In this review paper, we present the state-of-the-art in this growing vital field.

III. Stream Processing

Stream processing is the perfect platform to process streams of data or data from sensors. It is designed to process, analyze and react on real time streaming by using Continuous queries. Continuous queries are SQL type queries which operate over time and in some time windows. Stream analytics is most important part of stream processing. It is an ability to calculate statistical or mathematical analytics continuously in real time within the stream.

Stream processing solutions are designed to operate huge volume data on the fly with scalable and highly available architecture which make them suitable for analysis of data in real time. In comparison with traditional database in which data get physically stored first, indexed and then processed by queries, Stream processing uses the data which streams through the server. It can also communicate with external data sources, which help to incorporate data into application as per requirement, or update processed information to an external database.

The invention of the ‘live data mart’ is the latest researched development in stream processing. It provides a continuous query access to streaming data which collected in memory. From the perspective of business user, the data mart provides the live view of streaming data continuously. A live analysis projects slices, dices, and gathers data dynamically in response to business user’s action. This all happens in real time. The following figure shows the live data mart and architecture of Stream processing solution.

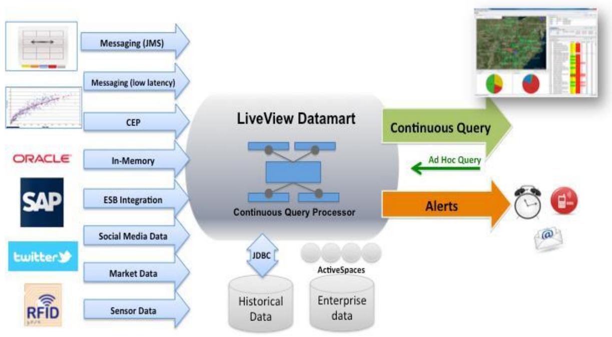

Figure 1: Stream Processing Architecture

The stream processing solution should have following key features.

- Real time responsiveness as the market conditions continually changes.
- Volume of data increases in size and complexity should not affect Performance and scalability.
- Processing massive amounts of streaming events like predict, monitor, act, automate, alert, aggregate etc.
- Fast updates along with existing data sources and infrastructure.
- Continuous query access to end user.
- Better analysis and visualization.

Use of Stream Processing in real world-

Stream processing was first used in the finance industry, as stock exchanges moved on to electronic trading. But today, wherever we are generating stream data through sensor or machine or human activity, it makes sense to use stream processing. In below mentioned industries, Stream processing can help to solve business challenges.

- E-commerce
- Intelligence and surveillance
- Pricing and analytics
- Data warehouse expansion
- Network monitoring
- Risk management
- Fraud detection

Following are some components which required to get above mentioned features and implement a stream processing[14].

Server: Low latency application server to process real time streaming.

IDE: Development environment which provides development, debugging and testing of stream processing process.

Connectors: To communicate with data sources like databases, Data warehousing, Market data, Statistics or technology, there should be pre-built data connectivity.

Streaming Analytics: UI to monitor, manage and analyze live stream data

Below are some widely adopted products which provides Stream processing solution and have above mentioned components-

- Apache Storm
- Apache Spark
- IBM InfoSphere Streams
- TIBCO StreamBase
- Apache Samza
- Esper
- AWS Kinesis
- DataTorrent
- Oracle CEP[13]

IV. Benefits of Stream Processing Solutions over Traditional database

The stream processing has so many benefits over the traditional relational database. Previously, the traditional coding was the only solution for the high volume streaming application. The traditional database has the data management system and were designed to handle static data. It works for online transaction processing (OLTP). As the large dataset has been stored into the database and user can initiate different types of queries, it creates an index for the database table before query processing and then user will get output. The storage of data, indexing and query processing makes traditional database unreliable for fast stream processing.

The applications which requires real time processing of large amount of data stream are making benefits over the limits of traditional database systems. There are some versions of traditional database which allows us to do data processing without going to the disk but again it cannot handle high speed streaming and large amount of data. The real time stream processing data comes from various applications such as IOT applications, sensors, etc. It is mainly used in military based applications because they want fast output. The real time stream processing introduces the processing of high speed data without going to the code. The software technologies like DBMS are also modifying the architecture to support the high speed data streaming. In traditional database, the custom code has been used to do high volume stream processing but it was not flexible. It has to go disk and get the data so it had slow response time.

The stream processing has so many advantages over the traditional. In stream processing, the data keeps going in a form of message passing. It doesn’t require any log records as traditional database. It doesn’t have requirement or not necessary to go to disk and write log records. The log records need so many I/O operations because it has to access the disk. It makes system slower. Therefore, in stream processing it doesn’t have log records instead the message passed as a stream. To process the data, we need some query mechanism to get the output or to get real time analysis. For streaming, the C++ and java has been used for the programming tools. Depending on low level programming result in the high maintenance cost and long development process. To replace these, the SQL is the high level language for the real time data. It is well known for almost all DBMS programmers. It gives the efficiency for the user generated queries. The other challenge with the stream processing is it needs the output without going to the disk data. In traditional database, it accesses the data before execute the query. Due to this the stream processing should handle imperfect streams as well. If the query is taking too much time, then it should be ended. There should be time out for those queries which are taking indefinite time so application can proceed with the partial data.

The stream processing predicts the output ahead of the time. It saves the time and handles the fault tolerance conditions. This is the very important feature in the stream processing. When there is a fault, it predicts the output and calculates the result for the same input stream when it recovers. The output of the two should be same. It does guarantee about the data safety and availability same like traditional database. It supports the multi threading operation to take advantage of the multi-processor systems. It does both process and response simultaneously.

Nowadays, the technology is moving towards the stream processing. The self-driving cars and IOT is developing very fast. In future, we might have the hybrid system. In this paper we have discussed about the real time stream processing and the technologies used in it..

V. Stream Processing In A Big Data World

Processing big volumes of data is not only reason for increase in the demand of stream processing. Data should be processed faster, so that organization can react to change in the business conditions in real time. In this section, we will go through what stream processing is, its role in big data architecture with Hadoop and Data warehousing.

Big data is one of the most trending words in recent years. It can be defined with three Vs as Volume, Velocity and variety. It is not only about the volume of the data but also about the variety of the data and the Velocity in terms of time. A big data architecture made up of various parts. Mostly, substantial amounts of structured and semi-structured data are stored in Hadoop which describes the volume and variety. On the other hand, fast data requirements are achieved by Stream processing which shows Velocity and Variety.

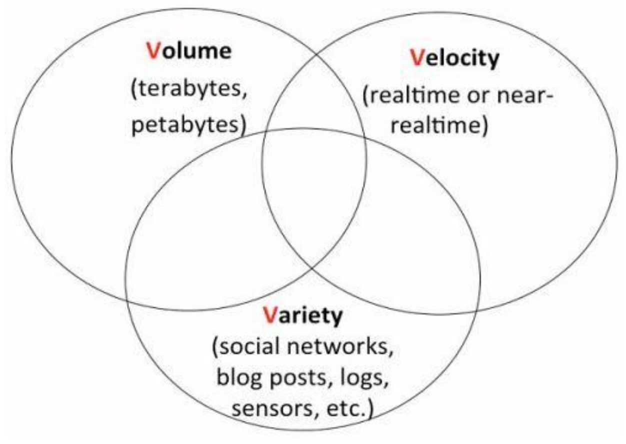
Volume
(terabytes, petabytes)
Velocity
(realtime or near-realtime)
Variety
(social networks, blog posts, logs, sensors, etc.)

Figure 2: Three Vs of Big Data

Relation of Stream Processing to Data Warehouse

Big data architecture consists of stream processing for real-time analytics. Hadoop is mainly used to store all type of data and long computations whereas the data warehouse stores the structured data for reporting and dashboards. All the three are very much different with each other. Hence, the integration of above technologies is more important and challenging as we need to combine more different sources and sinks.

A DWH can store terabytes of data and able to answers queries about historical data within seconds. However business may wants to know up-to-date information instead of using an approach where you may only get information on basis of historic data. This is where stream processing comes in and feeds all new data into the DWH immediately.

VI. SQL FOR STREAM PROCESSING

SQL as a language does not imply the need for structured data store. It can be used for real-time data processing as well. SQL enables real-time analytics to be expressed using simple queries. It encapsulates the underlying complexities of data management operations. SQL can be used for dynamic query planning and optimization. Streaming aggregation is a strength of SQL. There are two types. Tumbling window or sliding window concepts can be used to aggregate data.

Oracle Continuous Query Language (Oracle CQL) supports streaming data. It is scalable and supports a large number of queries over continuous data streams and traditional stored data sets. It uses Oracle CEP (Complex Event Processing) which is a Java server for the development of high-performance event driven applications. The Oracle CEP application is composed of Event Adapter, Input Channel, CQL Processor, Output Channel and EventBean. The Event Adapter provides event data to input channel. The CQL processor operates on these events and the query results are written to the output channel. The Event Bean takes actions based on the events it receives from the output channel.

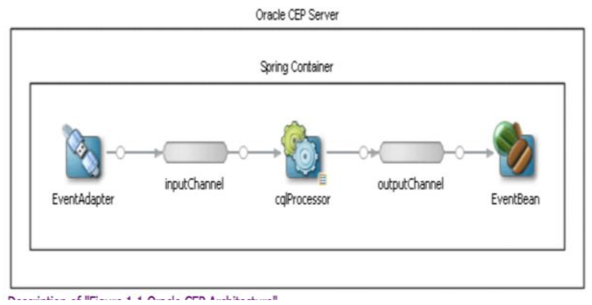

Figure 3: Oracle CEP Architecture

VII. TECHNOLOGIES USED IN REAL-TIME STREAM PROCESSING

Event data from web applications, mobile applications, medical devices, sensors need to be monitored as the stream of events occur. For real-time insights, it is really important to process the events. Technologies like Apache Kafka, Apache Spark and HBase can be used for this purpose.

Apache Kafka: Apache Kafka is an open-source stream processing platform which is used for building real-time data applications and stream applications. It is horizontally scalable, fast, fault-tolerant. It is used to build real-time streaming data pipelines that can reliably get data between the systems. Basically, Kafka runs as a cluster on one or more servers. It stores streams of records in categories called topics, where each record consists of a key, a value and a timestamp. Kafka has four core APIs. The Producer API allows an application to publish a stream of records to one or more Kafka topics. The Consumer API allows an application to subscribe to one or more topics. It allows to process the stream of records produced to them. It is possible to do simple processing using the producer and consumer APIs. But, for more complex transformations, Kafka provides Streams API. The Streams API allows an application to act as a stream processor. It consumes an input stream from one or more topics and produces an output stream to one or more output topics, and thus transforms the input streams to output streams. The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. Kafka is a special purpose distributed file system dedicated to high-performance, low-latency commit log storage, replication and propagation. The stream API builds on the core primitives Kafka provides i.e. it uses the producer and consumer APIs for input, uses Kafka for stateful storage and uses the same group mechanisms for fault tolerance among the stream processor instances. By combining storage and low-latency subscriptions, streaming applications can treat both past and future data the same way. That is a single application can process historical, stored data but rather than ending when it reaches the last record it can keep processing as future data arrives. This is a generalized notion of stream processing that subsumes batch processing as well as message-driven applications.

Apache Spark : Apache Spark is an open-source cluster-computing framework. Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark Streaming receives live input data streams and divides the data into batches. There are then processed by the Spark engine to generate the final stream of results in batches. The processed data can be pushed out of file systems, databases and live dashboards. Data sources such as HDFS Directories, TCP Sockets, Kafka, Flume, etc are supported by Spark Streaming.

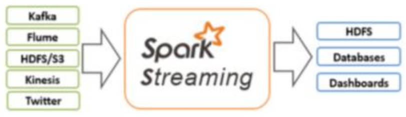

Figure 4 : Apache Spark Streaming

Spark Streaming divides the data stream into batches of X seconds called DStreams. DStream or Discretized Stream represents a continuous stream of data represented by a continuous series of RDDs, Resilient Distributed Datasets (Spark's abstraction of distributed dataset). The Spark Application processes the RDDs using Spark APIs and then the processed results are returned in batches.

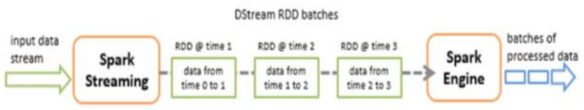

Figure 5: Apache Spark Streaming Architecture

Apache Hbase : Apache Hbase is a distributed non-relational datastore and is a source for reading RDDs and a destination for writing RDDs.

Let us consider an example use-case to understand how to integrate Apache Spark Streaming, MapR-DB and MapR streams.

(a) Data Source : Spark Streaming supports data sources such as Kafka, Flume, HDFS Directories. Let us consider MapR Streams which is a big data-scale streaming system. It enables clients and servers to exchange events in real time via Apache Kafka. Kafka topics organize events into categories. The topics are partitioned to make them scalable as that

(b) Process Data : Data can be processed with Spark's core APIs. Spark streaming divides data streams into Dstreams which is a sequence of RDDs. RDDs are spread across multiple nodes in the cluster. The date contained in RDDs is partitioned and operations are performed in parallel on the data.

(c) Store the data : The data store needs to support fast writes and scales for storing streaming data. WIth HBase API or MapR-Db, the table is partitioned across a cluster of key range and each server is the source for a subset of a table. Applications like Dashboard use the processed data.

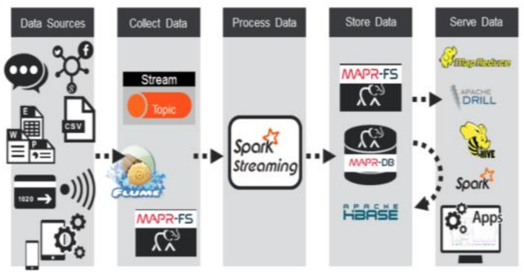

Figure 6 : Stages in Stream Processing

Apache Flink : Apache Flink is an open-source framework for distributed stream processing. Applications can maintain the summary of data that has been processed over time. It supports stream processing and windowing with event time semantics. Flink Program consists of Data Source which consists of the incoming data, Transformations which is the processing step and the data sink where the processed data is sent.

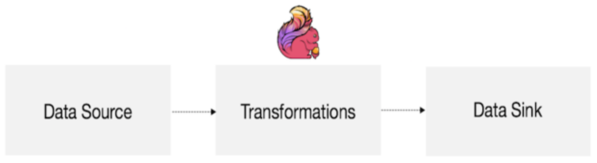

Figure 7: Apache Flink

VIII. SQLS : A STORM-BASED QUERY LANGUAGE SYSTEM FOR REAL-TIME STREAM DATA ANALYSIS

The early centralized data query streams such as stream or aurora have all operations performed on one node. When the resources reach saturation, data processing speed will decrease, output delay will increase, and the throughput will be reduced. Distributed data stream query systems where in computations are distributed across many computing nodes faced similar problems. Anyhow the performance of other nodes is affected when one of the node reaches resource saturation. Amongst various systems, storm has better performance in terms of real-time data stream processing. Usually, there are two types of queries, ad hoc query and continuous query.

Adhoc queries are the ones which search the unprocessed historic data within specific time periods whereas continuous queries continuously perform query processing over ever-incoming data. streams. Ad hoc query needs to search historic data, which demands data storage of the system. Therefore, most data stream query systems only support continuous query.

But this Storm-based query language System (SQLS), supports both continu-
ous queries and one-time queries over streaming data. There is another constraint which is of memory i.e the sliding window size has to be less than the main memory size and this constraint is not valid in many circumstances. Therefore storm based query language system uses redis memory database to expand storage for sliding window. A Stormed based distributed parallel computing architecture is employed in the SQLS system. And the join operator has been optimized by the use of cache and Redis. Therefore, the time delay of join operator has been greatly reduced.

SQLS system consists of the following :

[I] Statement parsing module: Statement parsing module consists of three components: lexical analysis, syntax analysis, and semantic analysis.Lexical analyzer parses the statement into keywords, recognizes keywords, and checks misspelling of keywords. Syntax analyzer conducts syntax analysis on the output of lexical analyzer, including the use of keywords, parameter passing, format matching. In the semantic analysis component, the implication of the statement is worked out to form a logical tree, forwarded to optimization module for optimization.

[II]Optimization module: Optimization module includes logic optimization, QoS optimization, and operator scheduling. Wherein the logic optimization is mainly for adjusting the sequence of operators and operator reuse.

[III]Operators repository module: Operations of the data are defined in this module, such as filter operator, projection operator, join operator, etc. Users can add customized operators.

[IV] Query module: Query module is divided into continuous query module and ad hoc query module.

[V]Data access module: The data access module has two sub-modules: socket forwarding module for continuous query, and disk storage module for ad hoc query. The data access module can be programmed to dynamically read various types of data streams according to user needs.

IX. APPLICATION FOR MONITORING STREAM - AURORA

Traditional database management systems are designed to address the needs of the application which does not require real time analysis of data. Firstly, the basic instinct on having the traditional DBMSs is that they have passive repository storing huge amount of data and then queries are fired on them. Second, it considers the historic data as well as current data. Based on these assumptions, one cannot work when the data is continuously incoming. There are many monitoring applications and one cannot work on them using traditional DBMSs. Monitoring applications are the one’s for which streams of information, triggers, imprecise data, and real-time requirements are prevalent. For example, military applications monitoring soldier locations. One cannot rely on stale data while monitoring such crucial application. Real time data is needed at every moment for making these applications effective. Plus the data stream is often lost and there is no historic data to work on therefore leading to approximation of the query answer. Also, most monitoring applications are trigger-oriented. If one is monitoring a chemical plant, then one wants to alert an operator if a sensor value gets too high or if another sensor value has recorded a value out of range more than twice in the last 24 hours. Every application could potentially monitor multiple streams of data, requesting alerts if complicated conditions are met. Thus, the scale of trigger processing required in this environment far exceeds that found in traditional DBMS applications. For these reasons, monitoring applications are difficult to implement using traditional DBMS technology. To deal with all this, aurora was conceptualized which is designed to better support monitoring applications.

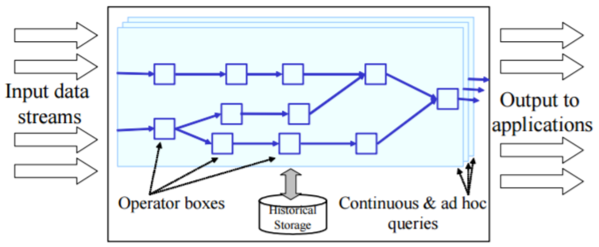

Figure 8 : Aurora System Model

Data that arrives at aurora is assumed to be imprecise and from variety of unreliable sources at irregular intervals. There is an aurora timestamp and unique identifier to every incoming data stream to monitor quality of service. The basic functioning of Aurora is that it processes incoming streams in the way that is defined by an application administrator. Aurora is fundamentally a data-flow system and uses the popular boxes and arrows paradigm found in most process flow and workflow systems. Hence, tuples flow through a loop-free, directed graph of processing operations (i.e., boxes). Ultimately, output streams are presented to applications, which must be programmed to deal with the asynchronous tuples in an output stream. Aurora can also maintain historical storage, primarily in order to support ad-hoc queries. Aurora can support eight primitive operations to support stream processing requirements. Few of them are - windowed operation : the window of few tuples are selected on the real time data and query is processed on them and then the window is slided down; filter operator, acts on single tuple, it checks if any tuple satisfies a specific predicate; join operator pairs tuples based on difference in timestamp; map operator applies to every tuple; drop operator drops random tuples at some rate specified as an operator input.

In the traditional relational query optimization technique, one of the primary objectives is to minimize the number of iterations over large data sets. On the other hand, Stream-oriented operators that constitute the Aurora network, are designed to operate in a data flow mode in which data elements are processed as they appear on the input. Although the amount of computation required by an operator to process a new element is usually quite small, it is expected to have a large number of boxes. Furthermore, high data rates add another dimension to the problem. Aurora uses : Dynamic Continuous Query Optimization and Ad-Hoc query optimization. In dynamic query optimization, the optimizer selects a portion of network to optimize. Then, it will find all connection points that surround the subnetwork to be optimized. It will hold all input messages at upstream connection points and drain the subnetwork of messages through all downstream connection points. To the identified subnetwork, the optimizer will then apply the local tactics which include inserting projects and combining boxes. Aurora processes ad-hoc queries in two steps by constructing two separate subnetworks. Each is attached to a connection point, so the optimizer can be run before the scheduler lets messages flow through the newly added subnetworks. The initial boxes in an ad-hoc query can pull information from the B-tree associated with the corresponding connection point(s). When the historical operation is finished, Aurora switches the implementation to the standard push-based data structures, and continues processing in the conventional fashion.

X. HYBRID BATCH AND STREAM PROCESSING

Traditionally, Batch and Stream Processing are two completely different streams with different ways to process data such as MapReduce for batch and Storm for streams. But, Modern Data Integration requires both batch and stream processing to be done by one software to support essential business processes. Apache Spark and Apache Flink are the recent data processing engines that provide models for both batch and stream processing.

In Batch Computing, an entire data set is accessed and then the transformation is performed. It is really important that the data processing engine has the ability to tolerate failure during long computations. The MapReduce run-time engine ensures that a computation will complete in the presence of network and node failures. This is achieved through re-execution of Map or Reduce Tasks. Spark also provides a programming model and run-time engine to tolerate failures. Spark allows for the re-execution and re-creation of RDDs by the ability to track the lineage of operations. Another approach to achieve fault tolerance is to save checkpoints from stream nodes to a persistent storage and restores from the checkpoint if a failure occurs. Aurora supports continual queries (real-time processing), views, and ad-hoc queries all using substantially the same mechanisms.

XI. Challenges of Data Streaming Processing

- The data size is very large and it becomes difficult to analyse it.
- The sub-challenge is to reduce the ambiguity and redundancy in the data.
- The processing tool should be able to utilise the cache so that the flow of data through the system is optimal.
- There are partitioning algorithms available to partition the application across the available resources but most of them are too slow to run dynamically.
- There may be noise disturbance in the incoming data.

XII. Conclusion

Stream processing is required when data has to be processed fast and / or continuously, i.e. reactions have to be computed and initiated in real time. This requirement is coming more and more into every vertical. Many different frameworks and products are available on the market already, however the number of mature solutions with good tools and commercial support is small today. In this report, we have discussed about Stream Processing and the need for real-time processing of huge amounts of data in recent time. We have discussed about the technologies that are used to process the stream data and why we need SQL type language for the Real-time Streaming analytics. A Storm-based Query Language is studied and the challenges faced in data stream processing has been mentioned.

XIII. Acknowledgement

We thank Ahmed Ezzat, Professor of Advanced Database Systems COEN 380, for the support and guidance throughout the project. We thank the members of other groups for their cooperation during the presentation of the project.

References

[1] Carol McDonald, "*Real-Time Streaming Data Pipelines with Apache APIs : Kafka, Spark Streaming and HBase*". Retrieved from https://mapr.com/blog/real-time-streaming-data-pipelines-apache-apis-kafka-spark-streaming-and-hbase/

[2] Oracle CEP CQL Language Reference. Retrieved from https://docs.oracle.com/cd/E16764_01/doc.1111/e12048/intro.htm

[3] Introduction to Flink, retrieved from https://flink.apache.org/introduction.html

[4] Federico Wachs, "*Why we need SQL for data stream processing ane real-time streaming analytics*", retrieved from http://sqlstream.com/2017/02/sql-for-real-time-streaming-analytics/

[5] Srinath Perara, " *Why we need SQL like Query Language for Realtime Streaming anayltics ",* retrieved from http://srinathsview.blogspot.co.uk/2015/02/why-we-need-sql-like-query-language-for.html

[6] Tathagata Das, Matei Zaharia and Patrick Wendell, "Diving into Apache Spark Streaming's Execution Model", retrieved from https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html

[7] Tore Risch, "Challenges of streaming data", retrieved from http://www.nesc.ac.uk/talks/1130/ToreRisch.pdf

[8] Mike Stonebraker, Ugur Cetintemel and Stanley Zdonik, "*The eight rules of real-time stream processing",* retrived from http://www.mondovisione.com/exchanges/handbook-articles/the-eight-rules-of-realtime-stream-processing/ and http://complexevents.com/wp-content/uploads/2006/07/StreamBaseEightRulesWhitepaper.pdf

[9] The Aurora Project at Brown University, retrieved from http://www.cs.brown.edu/research/aurora/

[10] D.Carney, et al., "Monitoring Streams – A New Class of Data Management Applications," VLDB 2002, retrieved from http://www.cs.brown.edu/research/aurora/vldb02.pdf

[11] M. Gaber, et al., "Mining Data Streams: A Review," SIGMOD 2005, retrieved from http://delivery.acm.org/10.1145/1090000/1083789/p18-5gaber.pdf?key1=1083789&key2=8510729421&coll=portal&dl=ACM&CFID=69980356&CFTOKEN=9423814

[12] E. Liarou, et al., "Exploiting the Power of Relational Databases for Efficient Stream Processing," EDBT '09, Proceedings of the 12th Int'l Conf on Extending Database Technology, retrived from http://delivery.acm.org/10.1145/1520000/1516398/p323-liarou.pdf?key1=1516398&key2=5799629421&coll=portal&dl=ACM&CFID=69980356&CFTOKEN=94238114

[13] Real-time Streaming Analytics, Reteived form http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/complex-event-processing-088095.html

[14] Kai Wähner, "Real-Time Stream Processing as

Game Changer in a Big Data World with Hadoop and Data Warehouse“, Retrieved from https://www.infoq.com/articles/stream-processing-hadoop