International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

Lightning CEP - Joining on High Velocity Stream

Vikas Kale[1], Kishor Shedge[2]

[1]Sir Visvesvaraya Institute of Technology, Chincholi, Nashik, 422101, India
[2]Sir Visvesvaraya Institute of Technology, Chincholi, Nashik, 422101, India

Abstract: ***With Internet of Things number of users and devices connected to internet is growing exponentially. Stream processing and CEP systems are designed to support class of applications which requires fast and timely analysis of high volume data streams. Complex event processing, or CEP, is event/stream processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances. In this paper we describe how we implemented join operation on high velocity streams for Lightning- High Performance & Low Latency Complex Event Processor. We will discuss how inverted index and other methods can help in improving performance of system.***

Keywords: CEP, Complex Event Processing, Stream Processing

1. Introduction

With Internet of Things number of users and devices connected to internet is growing exponentially. Each connected device and its user generates lot of data which organizations want to analyze and use for their businesses. Generated data can be system logs, user activity, sensor reading and transactions in financial systems etc. Traditional systems stores data in RDMS and they use query language like SQL to retrieve required data for business use. BIG or internet scale data is very high volume of data which is in Peta Bytes. Database systems are not able to handle BIG data so Hadoop like batch processing are evolved to process BIG data and they provide offline data processing capability.

Stream processing systems support a different class of applications which continuously consumes and process data while continuously producing results. Touple is input data element and a continuous flow of touples is called streams. Examples of streams include user-click, event logs, network traffic, readings from sensor (GPS location, traffic movement, temperature), and various other data feeds. Stream processing systems are used to provide content to user and help organization to make better and faster decisions. Users of the content based system want's real-time information about surrounding e.g. news. Enterprise or organization wants real-time information from their system to detect intrusion, analyze fraud, analyze social media trends etc. Many open source and commercial stream processing systems are evolved and they provide basic infrastructure for stream processing.

Stream processing applications have very different requirements than those of batch processing applications. Order of data receipt impacts result of system. So stream processing applications are temporally sensitive. So they are generally time-critical because their use is promptness with which results are produced. Systems which find network intrusions or credit cards fraud patterns should respond quickly to an observed threat. Some other examples of stream applications include real-time video processing, automated stock trading, geo-spatial trajectory modification and vital-signs monitoring. Results produced by such applications are often urgent and they require immediate action. Importance and applicability rapidly decrease if result of these applications becomes more and more delayed. Best example of this is intrusion detection systems. If organization is able to identify intrusion detection and its patterns they will be able to defend their system against attack. In financial system like stock trading if firm is able to identify stock trends before others they will be able to gain more profit. In time-critical stream processing, it is important to minimize the average latency of the continuously emitted results instead of throughput. While stream applications which are not time-critical process as large a stream as possible with maximizing throughput.

Complex event processing, or CEP, is subset of event/stream processing which combines data from multiple sources. CEP infers events or patterns from multiple sources that suggest more complicated circumstances. While Stream processing system consider each event separately, CEP systems consider complex event patterns that considers the multiple and related events. Aim of this papers to describe how we designed Join processing elements which are critical to CEP.

2. Architecture

CEP combines data from multiple sources and finds events or patterns that related to complicated circumstances. Main difference between traditional DBMS and CEP systems is tradition DBMS system store data and then process then using queries. CPE systems do not store data they processes data as it is available. Following Figure 1 shows difference between CEP system and database.

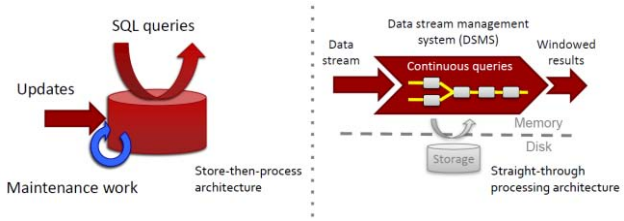

Figure 1: DBMS VS CEP Engine

Figure 2 show architecture of Lightening CEP. It uses pull model which pulls data from different sources and CEP engine process that data. It will also use lock free data structure for inter thread communication which is very fast compared to traditional Queue.

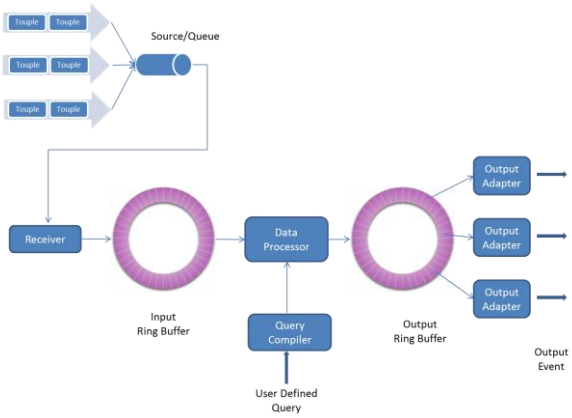

Figure 2: Lightning CEP Architecture

Input to CEP engine is Input adapter or receiver. It reads events from event source and sends them to core engine for processing. User can define custom Input Adapter which can pull data from multiple sources like web feeds, user clicks, queues, and log file etc.

Ring Buffer which provides lock free inter-thread communication is used for communication between different threads. Multiple buffers can be connected to each other to form processing pipeline as shown in Figure 2 has input and output ring buffers.

Queries can be defined by user as per requirement. Query compiler parses queries provided by user and required request processing pipeline of processing objects is created. Query pipeline is like directed graph of Ring Buffers. On each buffer multiple threads operates and data processing is done. Processors do work on input tuples like aggregation, window operations, filtering.

Output buffer receives events once events are processed by processing engine. One or more output adapter threads operate on output events from buffer and do required processing.

DSL is used to provide query capability. Internal DSL is implemented as Fluent Interface. Following is example of Fluent Interface.

```
SelectQuery query = new SelectQuery("StreamID");
query
 .where("Code").equal("IBM")
 .and("Price").greaterThan(500);
```

3. Join Queries

This paper specifically focuses on combining or joining two streams. For more details of architecture please refer previous papers on Lightening CEP.

3.1 Join Queries

User can define simple Join queries on stream. Join queries are defined on two different streams with some common data. Blow code query illustrates a JOIN of two data streams. First stream is for stock orders, and second is for the resulting stock trades. Output of query is a stream containing all Orders matched by a Trade within one second of the Order being placed. The output stream is sorted by timestamp, in this case, the timestamp from the Orders stream.

```
FROM Orders
   window 1 sec
JOIN Trades
ON Orders.orderId = Trades.orderId
   SELECT Orders.TimeStamp, Orders.orderId,
Orders.ticker,
    Orders.amount, Trade.amount
into DataStream
```

3.2 Building Query Pipeline

Building query pipeline for Join requires two considerations. First is how streams are joined and second consideration is how events are stored for expiry. Traditional CEP systems used Queues for inter-thread communication. Queues easiest data structure available out of box their performance degrades under heavy load. They also traditional data store like database creates bottleneck as they are slow.

Lightening uses multiple data structure for query join. It builds inverted index of events in memory. When event in stream arrives it is matched against inverted index of other stream and output event is generated. Next session discuss detailed design of Join pipeline.

4. Design

Figure 3 shows query pipline for Join Query for two streams.

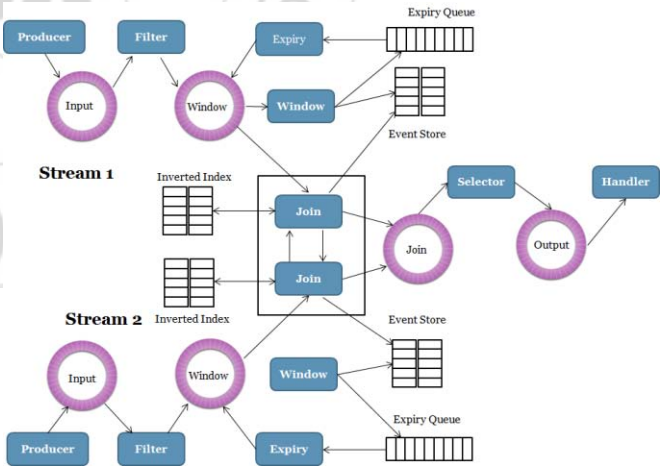

Figure 3: Join Query Pipeline

We will discuss main components in details in following sub sections

4.1 Input and Filter

Producer pulls events from source and put is input junction. Filter filters events based on filter condition in query. If condition matches events are sent to Window junction.

4.2 Window

Window Processor persist events to Event Store for future use. Event store is low latency store which can read and write millions of events per second. Event identifier is also put in Expiry Queue. Expiry queue is lock fee queue which holds events for expiry.

Expiry Processor pulls event ID from Expiry Queue and checks it for expiry. If event is expired it added to window junction and removed from event store.

4.3 Join

Join processor joins two streams when event arrives. Figure 4 shows details of join processor.

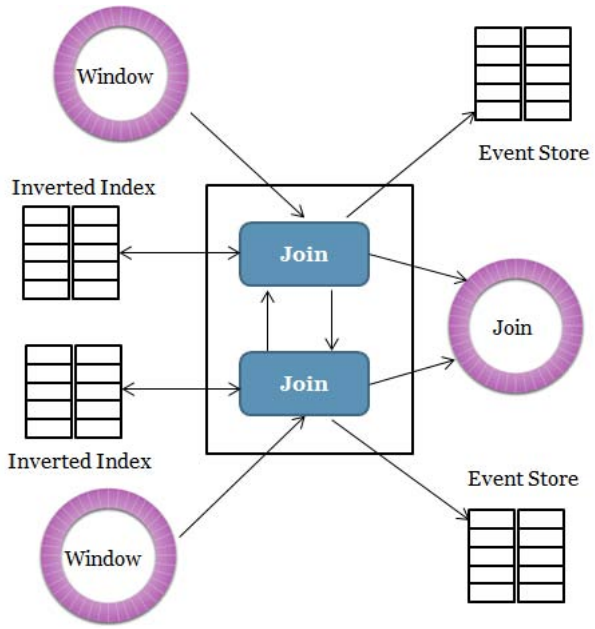

Figure 4: Two stage pipeline

Join Processor from each stream maintains inverted index for all events. Inverted index maintains ID's for all events for each value. When new invent arrives Join Processor for Stream 1 it adds ID of event in index. This new event is sent to Join Processor from Stream 2 which joins events with all matching events in its index and generate output event in Join junction.
For joining events all ID's are retrieved from inverted index based on join value. For each ID, event is retrieved from Event Store and join event is created.

4.3 Window

Following example shows how inverted index is used.

Stream definition is shown below and join query is executed on Code and Price columns.

StockStream {Code, Price, Quanity}

Table 1 shows inverted index for Code column and Table 2 shows inverted index for Price column.

Table 1: Code Inverted Index

Value	Event ID
IBM	23, 53, 35, 67, 34,..
Infosys	34, 76, 98, 56, 89

Table 2: Price Inverted Index

Value	Event ID
200	23, 76, 98, 67, 34,..
300	34, 53, 35, 56, 89,..

For joining events from two streams, index for each value is retrieved and common events from multiple indexes are selected.

When following events arrives in stream, indexed ID's for Code "IBM" and indexed ID's for price "200" are retrieved.

StockEvent {IBM, 200, 50}

Values from each index is retrieved

IBM Index ID's: 23, 53, 35, 67, 34
200 Index ID's: 23, 76, 98, 67, 34

Intersection of two indexes is matching events

Join Event ID's: 23, 67, 34

5. Conclusion and Future Work

In this paper we have discussed how system can be designed for joining events from two streams. Inverted index provides efficient way of joining two streams as events are not retrieved from Event Store.

When we have multiple join column finding intersection of multiple arrays is big task when we have millions of events. We can offload this work to GPU which can execute data parallel algorithm and give faster result.

References

[1] Vikas Kale, Kishor Shedge , Lightning CEP - High Performance & Low Latency Complex Event Processor – Volume 3 Issue 11 November 2014
[2] David Luckham & Roy Schulte, Event Processing Glossary – Version 2.0 , http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2/, [Online; accessed on 12/9/2014]
[3] David Luckham & Roy Schulte, Event Processing Glossary – Version 2.0 , http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2/, [Online; accessed on 12/9/2014]
[4] Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Google, Inc.

[5] Storm Distributed and fault-tolerant realtime computation. https://storm.apache.org/, [Online; accessed on 12/9/2014]
[6] S4 distributed stream computing platform. URL http://incubator.apache.org/s4/, [Online; accessed on 12/9/2014]
[7] Apache Samza is a distributed stream processing framework. http://samza.incubator.apache.org/, [Online; accessed on 12/9/2014]
[8] Drools Business Rules Management System (BRMS). http://www.drools.org/, [Online; accessed on 12/9/2014]
[9] Daniel J. Abadi, Don Carney, et al, Aurora: a new model and architecture for data stream management. Springer-Verlag 2003
[10] Understanding Java Garbage Collection. http://www.cubrid.org/blog/dev-platform/understanding-java-garbage-collection/, [Online; accessed on 12/9/2014]
[11] Reducing Garbage-Collection Pause Time. http://javabook.compuware.com/content/memory/reduce-garbage-collection-pause-time.aspx, [Online; accessed on 12/9/2014]
[12] Controlling GC pauses with the GarbageFirst Collector. http://blog.mgm-tp.com/2014/04/controlling-gc-pauses-with-g1-collector/, [Online; accessed on 12/9/2014]
[13] How to tame java GC pauses? Surviving 16GiB heap and greater. http://java.dzone.com/articles/how-tame-java-gc-pauses, [Online; accessed on 12/9/2014]
[14] Understanding GC pauses in JVM, HotSpot's minor GC. http://blog.ragozin.info/2011/06/understanding-gc-pauses-in-jvm-hotspots.html Accessed on [Online; accessed on 12/9/2014]
[15] Trisha Gee & Michael Barker / LMAX , The Disruptor - A Beginners Guide to Hardcore Concurrency, JAX conference 2011 London [Online; accessed on 12/9/2014]
[16] Trisha Gee, Dissecting the Disruptor: What's so special about a ring buffer? http://jaa.dzone.com/articles/dissecting-disruptor-whats-so, [Online; accessed on 12/9/2014]
[17] Martin Fowler, The LMAX Architecture. http://martinfowler.com/articles/lmax.html , [Online; accessed on 12/9/2014]
[18] Daniel J. Abadi, Yanif Ahmad, et al. The Design of the Borealis Stream Processing Engine. 2005 CIDR Conference
[19] D. Abadi, D. Carney, U. Cetintemel, et al. Aurora: A Data Stream Management System. 2003 ACM
[20] Terence Parr, The Definitive ANTLR Reference. The Pragmatic Programmer Publication ISBN-10: 0-9787392-5-6
[21] Esper Reference, By Esper Team and EsperTech Inc.
[22] Arun Mathew, Benchmarking of Complex Event Processing Engine – Esper.
[23] Leonardo Neumeyer, Bruce Robbins, Anish Nair, Anand Kesari Yahoo Labs. S4: Distributed Stream Computing Platform. 2010 IEEE International Conference on Data Mining Workshops
[24] Sriskandarajah Suhothayan, Isuru Loku Narangoda, Subash Chaturanga. Siddhi: A Second Look at Complex Event Processing Architectures. November 2011 ACM 978-1-4503-1123-6/11/11
[25] Kristian A. Nagy, Distributing Complex Event Detection. June 2012
[26] D. Abadi, Y. Ahmad, et al. The design of the borealis stream processing engine. In Second Biennial Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA, pages 277–289, 2005.
[27] D. Abadi, D. Carney, et al. Aurora: a data stream management system. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pages 666–666, 2003.
[28] M. Aguilera, R. Strom, et al. Matching events in a content-based subscription system. In Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, pages 53–61, 1999.
[29] D. Arvind, A. Arasu, et al. STREAM: the stanford stream data manager. In IEEE Data Engineering Bulletin, 2003.
[30] Aurora project page. http://www.cs.brown.edu/research/aurora/. [Online; accessed on 12/9/2014]
[31] The borealis project. http://www.cs.brown.edu/research/borealis/ public/. [Online; accessed on 12/9/2014]
[32] M. Cammert, C. Heinz, et al. Pipes: A multi-threaded publish-subscribe architecture for continuous queries over streaming data sources. Technical report, Citeseer, 2003.
[33] S. Chandrasekaran, O. Cooper, et al. TelegraphCQ: continuous dataflow processing. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pages 668–668, 2003.
[34] GIANPAOLO CUGOLA and ALESSANDRO MARGARA, Processing Flows of Information: From Data Stream to Complex Event Processing. ACM Computing Surveys, 2011
[35] S. Rizvi. Complex event processing beyond active databases: Streams and uncertainties. Master's thesis, EECS Department, University of California, Berkeley, Dec 2005
[36] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pages 407–418, New York, NY, USA, 2006. ACM. URL: http://doi.acm.org/10.1145/1142473.1142520.
[37] Martin Thompson, Dave Farley, Michael Barker, Patricia Gee, Andrew Stewart. Disruptor High performance alternative to bounded queues for exchanging data between concurrent threads. May-2011

Author Profile

Vikas Kale received the B.E. degree in Electronics Engineering from Pune University in 2005. He now studies M.E. computes in Pune University.