International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Lightning CEP - High Performance & Low Latency Complex Event Processor

Vikas Kale[1], Kishor Shedge[2]

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

Abstract: ***Number of users and devices connected to internet is growing exponentially. Each of these device and user generate lot of data which organizations want to analyze and use. Hadoop like batch processing are evolved to process such big data. Batch processing systems provide offline data processing capability. There are many businesses which requires real-time or near real-time processing of data for faster decision making. Hadoop and batch processing system are not suitable for this. Stream processing 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 I propose "Lightning - High Performance & Low Latency Complex Event Processor" engine. Lightning is based on open source stream processor WSO2 Siddhi. Lightning retains most of API and Query Processing of WSO2 Siddhi. WSO2 Siddhi core is modified using low latency technique such as Ring Buffer, off heap data store and other techniques.***

Keywords: CEP, Complex Event Processing, Stream Processing.

1. Introduction

Number of users and devices connected to internet is growing exponentially. Each of these device and user generate lot of data which organizations want to analyze and use for their businesses. Data can be user activity, system logs, and transactions in financial systems etc. Traditional systems stores data in RDMS and uses SQL query language to retrieve required data for business use. BIG data is high volume of data which is in scale of Peta Bytes. Transitional RDBMS systems are not able to handle BIG data so Hadoop like batch processing are evolved to process BIG data. Batch processing systems provide offline data processing capability.

Stream processing frameworks support a different class of applications. These are applications which continuously consume and process data while continuously producing results. Data element is called touple and a continuous flow of touples is called streams. Examples of streams include event logs, user-click, network traffic, readings from sensor (temperature, GPS location, traffic movement), and various other data feeds. Stream processing frameworks provide content to user and help organization to make better and faster decisions. User want's real-time information about surrounding like news. Organization wants real-time information from their system to analyze fraud, detect intrusion, analyze social media trends etc. Over time many open source and commercial stream processing systems are evolved and they provide basic infrastructure for stream processing.

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

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. Stream processing system consider each event separately while CEP systems consider complex event patterns that considers the multiple and related events.

Aim of this papers to represent "Lightning - High Performance & Low Latency Complex Event Processor" engine. Lightning is based on open source stream processor WSO2 Siddhi. WSO2 Siddhi is based on pipeline architecture which uses Queue. Lightning retains most of API and Query Processing of WSO2 Siddhi. WSO2 Siddhi core is modified using low latency technique such as Ring Buffer, off heap data store and other techniques.

2. Study of Existing CEP

Complex event processing, or CEP, is event or stream processing which combines data from multiple sources to find events or patterns that suggest more complicated circumstances. Following is difference between traditional

DBMS and CEP systems. Tradition DBMS system store data before processing CPE systems processes data as it is available. Figure 1 shows comparison between database and CEP system.

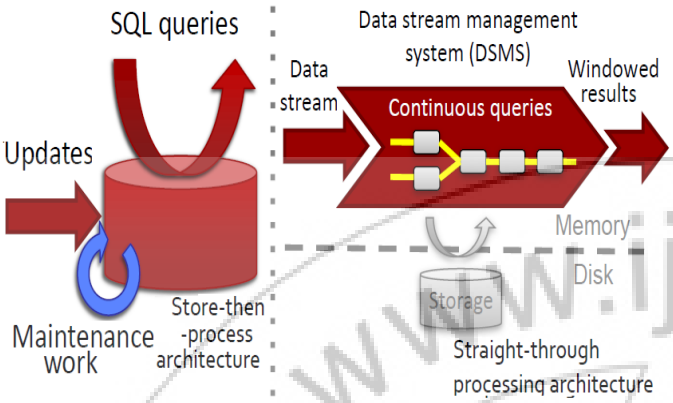

Figure 1: DBMS VS CEP Engine

Kristian[24] and GIANPAOLO[33] provide details CEP engine. We will review work done in complex event processing. Rizvi[34] surveyed complex event processing work in multiple contexts, active databases among others. It was concluded that processing continuous stream data by storing in data base and executing queries it resulted in very poor performance of systems. He extended TelegraphCQ data stream processor to process execute complex event and execute continuous queries (CQs), with traditional SQL-like continuous queries. With more result stream databases immerged but they used RDMS approach by storing data in file system which were not optimized for sequential streaming queries. The TelegraphCQ bas based on PostgreSQL architecture and supported continuous queries over streaming data, and later Truviso provided deep integration with databases system and supporting historical queries. But these systems failed to provide expected speed required of high volume CEP system.

Diao [35] presents the design of system called SASE which executes complex queries over event streams over real time RFID reading as events. They proposed new complex event processing language which allows queries to filter and correlate events. It also transforms the relevant events into new composite events which will form output. An event query provides features such as parameterization, sequencing, windowing and negation required for emerging RFID event monitoring applications. Effectiveness of SASE was demonstrated in a detailed performance study. Also SASE was compared to TelegraphCQ a relational stream processor.

There are many stream processing systems developed which followed pull or publish subscribe approach. Many also provide SQL like query language using which user can define queries. User can define different filters and windows for data aggregation. They have compromised ease of use over performance. Some of these systems are Aurora [7], Borealis [17] and S4 [22].

Recently developed Twitter Storm, S4 and Samza are excellent in processing high velocity streams but they are not CEP system. They all provide very low level API and rule engine should be built by user manually. They do not provide system to express queries which can relate different events so we cannot categorize them as CEP systems.

More recently new stream processing engines like SASE, WSO2 Siddhi, Esper are emerged which provides full complex event processing capability. Also there are many commercial systems emerged for stream processing which are based in existing open source application.

We found Siddhi and Esper provides excellent functionality for Complex event processing. While Esper has restricted some features in commercial license Siddhi is fully open source application.

While Siddhi is very good CEP engine it is based on queue architecture [23]. It uses java queues to create query processing pipeline and inter-thread communication. Queues easiest data structure available out of box their performance degrades under heavy load [36].

In order to put some data on a queue, you need to write to that queue. Also to take data out of the queue, you need to modify/write to the queue to removal the required data. This is write contention - where we have more than one client may need to write (add or remove) to the same data structure. Process requests from multiple clients a queue often uses locks. When a lock is used, it can cause a context switch to the kernel. When context switch happens the processor involved is likely to lose the data in its caches.

Table 1: Comparative throughput (in ops per sec)

	Array Blocking Queue (ns)	*Disruptor (ns)*
Min Latency	145	29
Mean Latency	32,757	52
99% observations less than	2,097,152	128
99.99% observations less than	4,194,304	8,192
Max Latency	5,069,086	175,567

Thompson [36] proposed new data structure based on ring buffer called Disruptor. The Disruptor has a lower concurrency overhead and significantly less write contention. It is also more cache friendly than other similar approaches. All these features results in greater throughput with lower latency. On processors at with average clock speed it can process over 25 million messages per second and over all latencies much lower than 50 nanoseconds. Disruptor has performance very high improvement over any other type of data structure like queues. It is very close to the theoretical limit of a modern processor to exchange data between cores.

Table 1 compares throughput i.e. operations per second between array blocking queue and Disruptor. Disruptor gives higher throughput than Queue.

3. Lightning Architecture

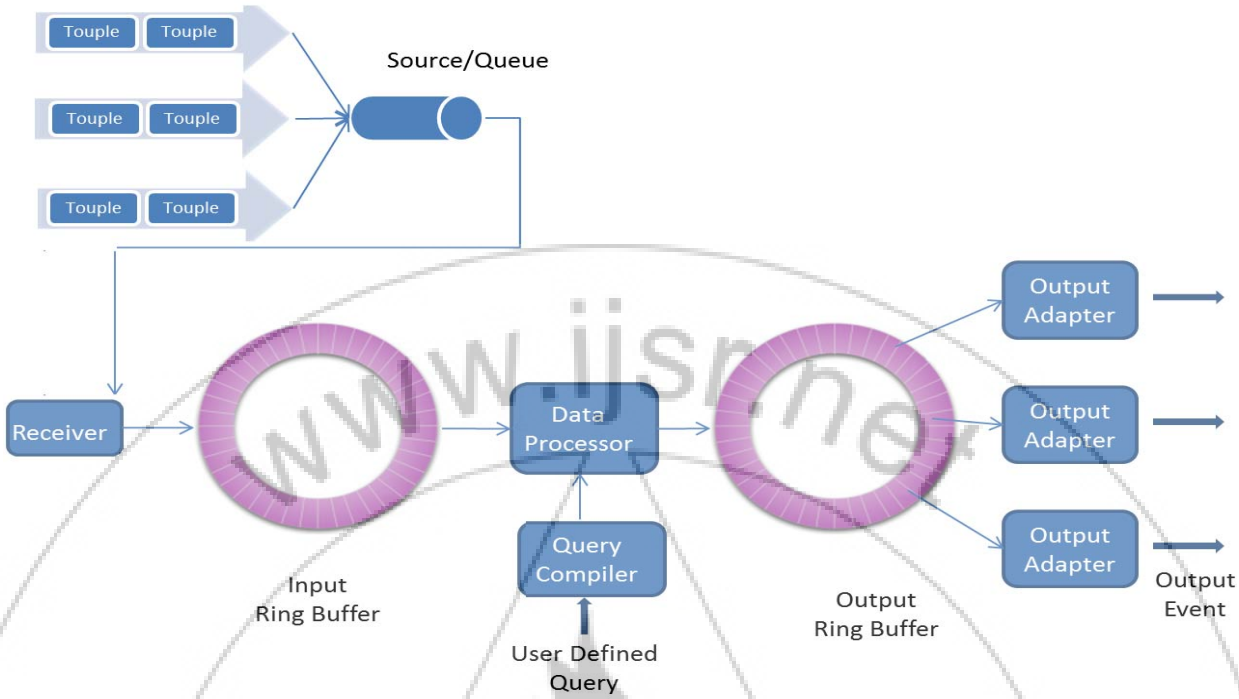

Figure 2: Lightning CEP Architecture

3.1 Architecture

Proposed "Lightning CEP" Architecture is shown in Figure 2. Lightning will use pull model which pulls data from different sources and send it to CEP engine. It will also use Disruptor for inter thread communication which is very fast compared to traditional Queue.

3.2 Source and Receiver

Lightening works on Pull architecture instead of push. Receiver can read data from multiple sources and send event to core engine. Receiver can read data from multiple sources like file system, web feeds, queues etc. Input event data can be in different format, so receiver will help in converting data to correct internal format.

3.3 Ring Buffer

Lightening uses Ring Buffer for inter thread communication. There can multiple ring buffers in system and above architecture shows input and output ring buffers. Receiver reads data from sources and sends it to input ring buffer for processing.

In input ring buffer there can be multiple threads working on buffer simultaneously. They will work on Tuple one or more at a time. Each thread does processing for one tuple and moves to other tuple.

Output ring buffer is same as input ring buffer. Once processing is done results are written to output ring buffer. Output adapter threads will process each element in buffer and produces requires output.

3.4 Data Processing Engine

Between input and out ring buffer is core of Lightening. User defines queries through admin interface. Processing Engine parses queries and creates requires request processing pipeline. Pipeline is directed graph of Ring Buffers which on which multiple threads operates and does data processing. It does all the work on input tuples like filtering, aggregation, window operations.

3.5 Output Adapters

Once processing engine is done with processing of data they emit output events to output ring buffer. Output adapter threads operate on output events from buffer and do required processing.

3.6 Query Compiler

Query compiler takes input query from user and compiles it. It parses and validates queries and create object graph which Processing Engine can understand. Query compiler is based in ANTLR and it's based on Siddhi query compiler which borrows all of the query definitions from it. Query compiler is based on Siddhi compiler and it borrows most of Siddhi Query specification

4. Conclusion and Future Work

As we discussed in literature review Stream processing engines provides better performance for analyzing high velocity stream data. There are few open source stream processing engines available which provide Complex Event Processing capability. Many of them use old data structures like Queues for input and inter thread communication. While

queues are good data structure they do not provide low latency and high throughput. We have proposed architecture of "Lightning CEP - High Performance & Low Latency Complex Event Processor".

Future work includes designing of detailed architecture and implementing it in open source CEP engine.

References

[1] 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]
[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] Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Google, Inc.
[4] Storm Distributed and fault-tolerant realtime computation. https://storm.apache.org/, [Online; accessed on 12/9/2014]
[5] S4 distributed stream computing platform. URL http://incubator.apache.org/s4/, [Online; accessed on 12/9/2014]
[6] Apache Samza is a distributed stream processing framework. http://samza.incubator.apache.org/, [Online; accessed on 12/9/2014]
[7] Drools Business Rules Management System (BRMS). http://www.drools.org/, [Online; accessed on 12/9/2014]
[8] Daniel J. Abadi, Don Carney, et al, Aurora: a new model and architecture for data stream management. Springer-Verlag 2003
[9] Understanding Java Garbage Collection. http://www.cubrid.org/blog/dev-platform/understanding-java-garbage-collection/, [Online; accessed on 12/9/2014]
[10] Reducing Garbage-Collection Pause Time. http://javabook.compuware.com/content/memory/reduce-garbage-collection-pause-time.aspx, [Online; accessed on 12/9/2014]
[11] 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]
[12] 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]
[13] 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]
[14] Trisha Gee & Michael Barker / LMAX , The Disruptor - A Beginners Guide to Hardcore Concurrency, JAX conference 2011 London [Online; accessed on 12/9/2014]
[15] 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]
[16] Martin Fowler, The LMAX Architecture. http://martinfowler.com/articles/lmax.html , [Online; accessed on 12/9/2014]
[17] Daniel J. Abadi, Yanif Ahmad, et al . The Design of the Borealis Stream Processing Engine,. 2005 CIDR Conference
[18] D. Abadi, D. Carney, U. Cetintemel, et al. Aurora: A Data Stream Management System. 2003 ACM
[19] Terence Parr, The Definitive ANTLR Reference. The Pragmatic Programmer Publication ISBN-10: 0-9787392-5-6
[20] Esper Reference, By Esper Team and EsperTech Inc.
[21] Arun Mathew, Benchmarking of Complex Event Processing Engine – Esper.
[22] Leonardo Neumeyer, Bruce Robbins, Anish Nair, Anand Kesari Yahoo Labs. S4: Distributed Stream Computing Platform. 2010 IEEE International Conference on Data Mining Workshops
[23] 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
[24] Kristian A. Nagy, Distributing Complex Event Detection. June 2012
[25] 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.
[26] 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.
[27] 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.
[28] D. Arvind, A. Arasu, et al. STREAM: the stanford stream data manager. In IEEE Data Engineering Bulletin, 2003.
[29] Aurora project page. http://www.cs.brown.edu/research/aurora/. [Online; accessed on 12/9/2014]
[30] The borealis project. http://www.cs.brown.edu/research/borealis/ public/. [Online; accessed on 12/9/2014]
[31] M. Cammert, C. Heinz, et al. Pipes: A multi-threaded publish-subscribe architecture for continuous queries over streaming data sources. Technical report, Citeseer, 2003.
[32] 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.
[33] GIANPAOLO CUGOLA and ALESSANDRO MARGARA, Processing Flows of Information: From Data Stream to Complex Event Processing. ACM Computing Surveys, 2011
[34] S. Rizvi. Complex event processing beyond active databases: Streams and uncertainties. Master's thesis,

EECS Department, University of California, Berkeley, Dec 2005

[35] 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.

[36] 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.