Lightning CEP - High Performance & Low Latency Complex Event Processor

Vikas Kale¹, Kishor Shedge²

¹,²Sir 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 large 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 tuple and a continuous flow of tuples 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 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](image)

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 stream data 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 has 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>
<thead>
<tr>
<th></th>
<th>Array Blocking Queue (ns)</th>
<th>Disruptor (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Latency</td>
<td>145</td>
<td>29</td>
</tr>
<tr>
<td>Mean Latency</td>
<td>32,757</td>
<td>52</td>
</tr>
<tr>
<td>99% observations less than</td>
<td>2,097,152</td>
<td>128</td>
</tr>
<tr>
<td>99.99% observations less than</td>
<td>4,194,304</td>
<td>8,192</td>
</tr>
<tr>
<td>Max Latency</td>
<td>5,069,086</td>
<td>175,567</td>
</tr>
</tbody>
</table>

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

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

[20] Esper Reference, By Esper Team and EsperTech Inc.
[21] Arun Mathew, Benchmarking of Complex Event Processing Engine – Esper,
[34] S. Rizvi. Complex event processing beyond active databases: Streams and uncertainties. Master’s thesis,
EECS Department, University of California, Berkeley, Dec 2005


[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.