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

Lightning CEP - Designing Filter and Window Operations

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: ***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. There are many businesses which requires real-time or near real-time processing of data for faster decision making. 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 we describe how we can design filter and window operations for Lightning - High Performance & Low Latency Complex Event Processor.***

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 wants" 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 can design Stream Processing system. In this paper we focus on how we can design Filter and Window processing elements which are critical to CEP.

2. Architecture

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.

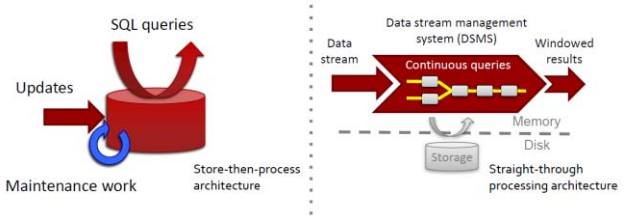

Figure 1: DBMS VS CEP Engine

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.

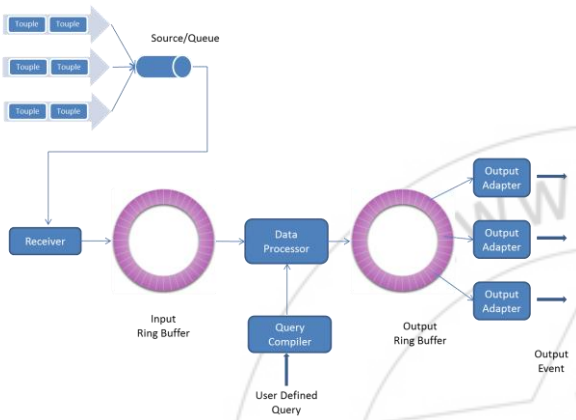

Figure 2: Lightning CEP Architecture

Lightning works on Pull architecture instead of push as used by other CEP like Siddhi or Esper. Input adapter or receiver can read data from multiple sources. Data is event and it is sent to CEP core engine for processing. It can read data from multiple sources like queues, web feeds, file system etc.

Ring Buffer is used for inter thread communication. Multiple ring buffers can be present in system and architecture shown in Figure 2 has input and output ring buffers.

Event processing logic resides between input and out ring buffer. User can define queries using admin interface. Query processing engine parses queries and creates required request processing hierarchy of objects like pipeline. Each pipeline is like directed graph of Ring Buffers on which multiple threads operates and does data processing. Operations performed on input tuples are like aggregation, window operations, filtering.

After processing engine is done with processing of data it emits output events to output ring buffer. One or more output adapter threads operate on output events from buffer and do required processing.

Lightening uses internal DSL to provide query capability. It implements Internal DSL as Fluent Interface. Following is example of Fluent Interface.

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

3. Design

3.1 Stream

Touple is input data element and a continuous flow of touples is called streams. Since stream processing is processing data before storage events are not stored unless it is required. Following is example of StockStream stream. Each touple in Stream will has following properties.

StockStream {Code, Price, Quanity}

When streams are defines user can define event source for queries. Event source pull data from external systems and puts in CEP.

3.2 Stream and Queries

User can define simple Filter, Window and Join queries on stream. Filter queries can filter events based on different conditions. Following is example of simple filter condition on Stock Stream

```
from StockStream where Code="IBM" and Price>500
Select *
Insert into StockPickStream
```

User can also define window queries which will help in aggregating events in small window if time.

```
from StockStream
  where code = „IBM" and price >500
     Window 0.5 sec
  Select Code, avg(price)
     Group by code
Select *
Insert into StockPickStream
```

3.3 Building Query Pipeline

Traditional CEP systems used Queues for inter-thread communication. Queues easiest data structure available out of box their performance degrades under heavy load.

In order to put some data on a queue, we need to write to that queue. Also to take data out of the queue, we 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.

Lightening uses Disruptor data structure for inter-thread communication. When user defines stream, stream junctions are created. As shown in Figure 3 Stream junctions are nothing but circular Ring Buffer where multiple threads can read and write without blocking other thread. It results in

lock fee operation where one thread can write events to junction while other thread can read and execute queries on events.

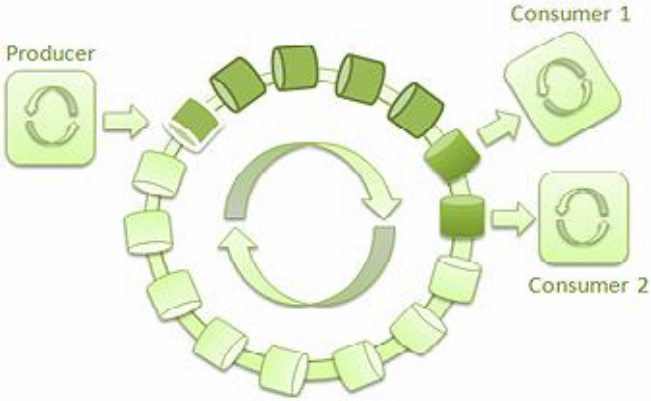

Figure 3: Single Producer multi consumer junction

Based on query defines by user system creates multiple junctions and assigns worker thread to each junction. Stream Processors executes logic based on user query and transfer events from one junction to other junction to create pipeline.

3.4 Filter Queries

We will see how filter queries are executed. Following is Select query on StockStream. It selects all events with code "IBM" and price greater than 500. All selected events are inserted in to stock pick stream.

```
from StockStream where Code="IBM" and Price>500
select *
Insert into StockPickStream
```

Figure 4 shows simple filter pipeline with filter and output selector.

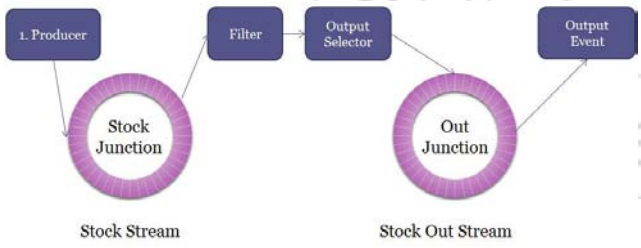

Figure 4: Simple Filter Pipeline

Query executor contains Filter and output selector. Filter executes where clause conditions. Output selector selects column specified in select statement.

In output junction system calls output event handler registered by user to generate real life event.

3.5 Multi Stage Pipeline

Single stage pipe line is fast and can process lots of events. To increase throughput we can create multi-stage pipeline. Multiple input junctions can be used to pull events from source and filter them. After filtering events both stages adds data to same output junction. Figure 5 show two stage pipeline.

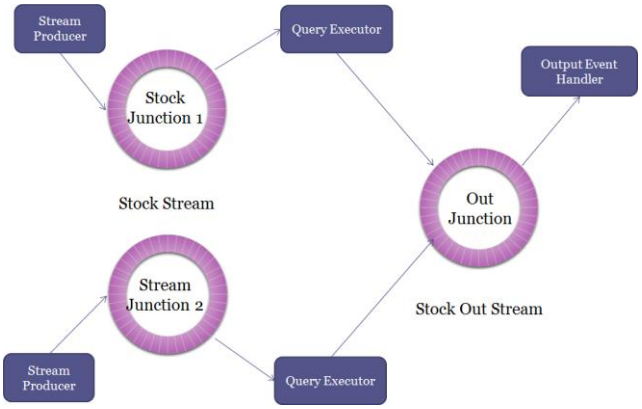

Figure 5: Two stage pipeline

We can almost double throughput by adding second stage in pipeline.

3.6 Multiple Filter Queries

User can define multiple filter queries on single stream. Lightening uses same input junction for two queries while create different output junctions. Figure 6 shows multi queries pipeline.

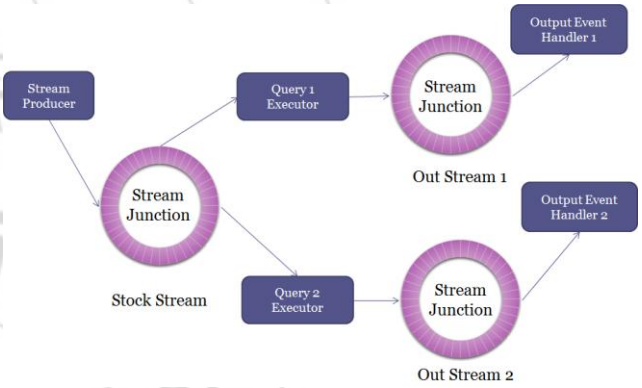

Figure 5: Multiple Queries on Same Stream

3.7 Window Queries

Window queries are used when user want to aggregates events for specific time window. Following window query on stock stream calculates average price of "IBM" stock over 0.5 sec.

```
from StockStream
   where code = „IBM" and price >500
      Window 0.5 sec
   Select Code, avg(price)
      Group by code
Select *
Insert into StockPickStream
```

Figure 7 show pipeline for window query. This pipeline uses three junctions. Producer puts data in input junction. Query executor executes filter query on input events and transfers matching events to Window Junction.

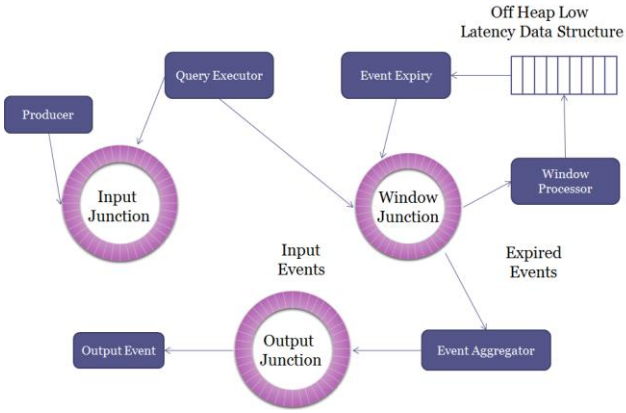

Figure 7: Window Query Execution

In window junction there are two processors. Window processor schedules events for expiry. Expiry events are placed in low latency off heap data structure like Chronicle, Ring Buffer or traditional data store like database based on expiry time. System allows user to select different data stores.

Event Aggregator aggregates received events. All events new and expired are processed by aggregator and it produces output events in output junction.

Performance of window operation depends on various factors like data store and expiry time. For high velocity streams if expiry time is short traditional data store like database creates bottleneck as they are slow. Putting and retrieving events from tradition data store takes time and it increases latency. Event Expiry producer picks expired events and put into Window Junction.

4. Implementation and Results

We have done initial implementation of proposed system. Basic architecture is implemented which created data processing pipeline for query processing. Current implementation includes Input and Output adapters, Query compiler and engine.

Benchmarking is done on Windows 7 machine running on "i7 2Ghz 2 Core 4 Thread" processor and 8GB RAM. Java version used is Java8.

Table 1: Comparative throughput (in ops per sec)

Pipeline Stage	Time Min	Events Processed
1 Stage	10	3-4 Million
2 Stage	10	7-8 Million

As we can see above table implemented system provides high throughput for filter queries.

5. Conclusion and Future Work

We have discussed how we can build stream processing engine with high throughput and low latency. This is just starting point of complete implementation of overall system. User can define input and output adapters. User can define filter and window queries over stream. Future work is to implement Join operation over window stream.

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.