STREAM-OPS: a Streaming Operator Library

Ricardo Jiménez-Peris[1], ValerioVianello[2] and Marta Patiño-Martínez[2]

[1] LeanXcale, Madrid, Spain
rjimenez@leanxcale.com
[2] Universidad Politécnica de Madrid, Madrid, Spain
{ vvianello,mpatino }@fi.upm.es

Abstract. Nowadays applications consume huge amount of live data with the requirement of real-time processing. Complex Event Processing (CEP) represents a promising technology to allow these applications to process large amount of information in real-time. Some of the available CEP systems, like Apache Storm, provide a programmatic model for the definition of the continuous queries used to process data on the fly. This paper presents STREAM-OPS, a library of streaming operators written in JAVA that is designed to ease the process of streaming query definition. In the paper we also present an evaluation of the STREAM-OPS library when integrated into two CEP systems developed in the context of CoherentPaaS and LeanBigData European projects.

1 Introduction

Nowadays applications consume huge amount of live data with the requirement of real-time processing. Complex Event Processing (CEP) represents a promising technology to allow these applications to process large amount of information in real-time. Some of the available CEP systems, like Apache Storm [1], provide a programmatic model for the definition of the continuous queries used to process data on the fly. This paper presents STREAM-OPS, a library of streaming operators written in JAVA that is designed to ease the process of streaming query definition.

In the last decade several implementations of CEP came out on the market from both academia and industry (Borealis, [4]), (Flink, [5]), (Spark, [6]). Among those, Apache Storm (Storm, [1]) is considered state of the art in distributed complex event processing. Storm is a distributed, reliable and fault-tolerant computation system currently released as an open source project by the Apache foundation. Storm does not provide a language for describing queries and operators on the streams. Everything is done programmatically. This approach although flexible, it is error prone and time consuming. In the context of CoherentPaaS [2] we developed a CEP engine built on top of Storm enriching it with several new features among which the incorporation of the STREAM-OPS library presented in this paper. STREAM-OPS library is also included in the engine developed in the context of LeanBigData [3] project that is a novel high scalable and efficient CEP. In the rest of the paper we first present the

operators available in the STREAM-OPS library and then we report the performance evaluation of some of these operators in a distributed setup.

2 Streaming Operators

Streaming queries are modelled as an acyclic graph where nodes are streaming operators and arrows are streams of events. Streaming operators are computational boxes that process events received over the incoming stream and produce output events on the outgoing streams. Algebraic operators can be either stateless or stateful, depending on whether they operate on the current event (tuple) or on a set of events (window).

Stateless operators process incoming events one by one. The output of these operators, if any, only depends on the last received event. Stateless operators provide basic processing functionalities such as filtering and projection transformations.

Stateful operators perform operations over a set of incoming events called sliding window. A sliding window is a volatile memory data structure. There are three types of sliding windows:

- Tuple-based window: it stores up to n tuples.
- Time-based window: it stores the tuples received in the last t seconds.
- Batch-based window: it stores all the tuples received between a start and stop conditions.

Tuple and time based windows must be configured with the size and advance parameters. The size parameter defines the capacity of the window (number of events/time in seconds) and the advance parameter defines which events must be removed from the window when the window is full.

STREAM-OPS library also provide a set of operators with the capability of reading and writing tuples from/to external data stores, these are called database operators.

2.1 Stateless Operators

The stateless operators in STREAM-OPS library are: map, filter, multi-filter and union. They are described in the following.

The **Map** operator it is a generalized projection operator defined as:

$$Map(S) = \{ A'1 = f1\ (t), A'2 = f2\ (t), \ldots, A'n = fn\ (t)\ , O\}$$

It requires one input stream and one output stream. The schema of these two streams may be different. The map operator transforms each tuple t on the input stream S by applying a boolean and/or arithmetic expression (fi). The resulting tuple have attributes A'1, . . . , A'n where, A'i = fi (t), and is sent through the output stream O.

The **Filter** is a selection operator defined as:

Filter(S) = {(P(t) , O)}

The filter operator requires one input stream and one output stream with the same schema. It verifies the match of tuples t on the input stream S with the user defined predicate P. When P(t) is satisfied the tuple t is emitted on the output stream O.

The **MultiFilter** is a selection and semantic routing operator defined as:

MultiFilter(S) = {(P1(t) , O1), (P2(t) , O2), . . . , (Pn(t) , On)}

The MultiFilter operator requires one input stream and at least one output stream, all with the same schema. The MultiFilter emits a tuple t on all the output streams Oi for which the user defined predicate, Pi(t) is satisfied.

The **Union** is a merger operator defined as:

Union(S1, S2, . . ., Sn){O}

The Union operator requires at least one input stream and only one output stream, all with the same schema. It is used to merge different input streams with the same schema into one output stream O.

Figure 1 shows an example of the Map operator. In the example the Map is used to transform the input tuples, with the schema [idcaller, idreceiver, duration, timestamp] representing a simplified Call Description Record (CDR) by adding a new field (cost) evaluated with the expression cost=duration*10 + 10.

Figure 1: Example of Map operator.

2.2 Stateful Operators

Two stateful operators are defined in STREAM-OPS library: aggregate and join.

The **Aggregate** operator computes aggregate functions (e.g., sum, average, min, count, ...) on a window of events. It is defined as:

Aggregate(S) = { A'1 = f1(t ,W), . . . , A'n = fn(t ,W) , s, adv, t, Group-by(A1, . . . , Am), O}

The aggregate operator accepts only one input stream and defines one output stream. It supports both time based sliding windows and tuple based sliding windows. Parameters s, adv and t define the size, the advance and the type of the sliding window. The Group-by parameter indicates how to cluster the input events; that is, the operator keeps a separate window for each of cluster defined by the attributes (A1, . . . , Am). Any time a new event t arrives on the input stream and the sliding window of the corresponding cluster is full, the set of aggregate functions $\{fi\}i1 \leq i \leq n$ are computed over the events in that sliding window W. The resulting tuple with attributes A'1, . . . , A'n where, A'i = fi(t ,W), is inserted in the output stream O. Finally, after producing the output tuple, all the windows are slid according with the advance adv parameter.

The **Join** operator joins events coming from two input streams. It is defined as:

Join(Sl,Sr) = {A'1 = f1(t ,Wl,Wr),.. , A'n = fn(t , Wl,Wr) , P, wl, wr, Group-by(A1, .., Am), O}

The join operator accepts two input streams and defines one output stream. Sl identifies the left input stream and Sr identifies the right input stream. P is a user defined predicate over pairs of events tl and tr belonging to input streams Sl and Sr, respectively; wl and wr define the size and the advance of the left and right sliding windows while de group-by defines the clustering as in the aggregate operator. In order to be deterministic the join operator only supports time based sliding windows. In the following we consider the simplified situation where the group-by parameter is empty and there is only one sliding window per stream. For each event tl received on the input stream Sl (respectively tr from stream Sr) the concatenation of events tl | ti is emitted on the output stream O, if these conditions are satisfied:

(1) ti is a tuple currently stored in Wr (respectively in Wl)
(2) P is satisfied for the pair tl and ti (respectively tr and ti)

The attributes A'1, . . . , A'n of tuples that are indeed inserted in the output stream O are a subset of the concatenation of events tl | ti where, A'i = fi(t ,Wl,Wr). After that all the output tuples triggered by the tuple tl (respectively tr) received on the input are produced, the sliding window Wr (respectively in Wl) is slid according with the advance parameter.

2.3 Database Operators

Database operators access the data stores using queries written in SQL. In STREAMOPS library there are available two database operators, ReadSQL to fetch data from the data store and UpdateSQL to store data in the data store.

The **ReadSQL** operator requires one input stream, S, and one output stream. The schema of these two streams may be different. The operator is configured with a parameterized query to be run against a data store. The parameterized query must be a SELECT statement. For each tuple, t, received on the input stream, S, the operator replaces the parameters in the query with the values read from the corresponding fields in the input tuple t and then, it executes the query. The operator produces as many tuples on the output stream as tuples has the result set of the query executed on the data store. Each output tuple is created either using fields of the incoming tuple, t, or fields of the result set row or a combination of them.

The **UpdateSQL** operator is in charge of storing results of the CEP query in a data store. It requires one input and one output stream. The schema of these two streams may be different. This operator is also configured with a parameterized query that must be an update statement, that is, it modifies, inserts or deletes data in the data store. For each tuple, t, received on the input stream S, the UpdateSQL operator replaces the parameters in the query with the values from the corresponding fields in the input tuple and then, it executes the query updating the data store. The UpdateSQL operator creates one output tuple for each input tuple. The output tuple can be either a copy of the input tuple or the number of modified rows in the data store or a concatenation of the two.

Figure 2 shows an example of the ReadSQL operator. The operator receives CDRs and fetches from an external data store the monthly plan (idplan) of the user making the phone call. The output tuple is composed by the fields idcaller, duration and timestamp of the input tuple plus the idplan field read from the data store.

Figure 2: Example of ReadSQL operator

3 Evaluation

This section reports the results of the evaluation of STREAM-OPS library when integrated into the two CEP systems developed in the context of CoherentPaaS and LeanBigData European projects.

We use two STREAM-OPS queries in the evaluation. A simple query, named *query1,* made be a single aggregate operator and a more complex query, named *query2*, composed with a filter operator followed by a map and an aggregate. The schema of the data used in the experiments is a simplified Call Description Record (CDR) schema composed by 4 fields labelled idcaller, idreceiver, duration and timestamp. The first two fields identify the caller and the receiver number of a phone-call. The field duration reports the duration of the phone-call in seconds and the field timestamp marks the time the phone-call was made.

In particular *query2* evaluates the average cost per user every 10 calls, taking into account that:

- Phone calls with duration shorter than 10 seconds are free.
- The cost of a phone call is 1 cent per second plus 10 cents per call establishment

In *query2* the filter is used to discard all phone calls with duration shorter than 10 seconds, the map is used to calculate the price of each phone call and the aggregate is used to calculate the average cost of each user. *query1* is a simplification of *query2* where we removed filter and map operators.

3.1 Deployment

The evaluation is being made at UPM cluster. We used several nodes each one equipped with a quad-core Intel Xeon X3220@2.40GHz, 8GB of RAM and 1Gbit Ethernet and a directly attached 160GB SDD disk running Ubuntu 12. To monitor the nodes used in the experiments we used the Ganglia Monitor System [7] that is a scalable distributed monitoring system for high-performance computing systems.

The setup is:

- 1 to 4 blades to run load generator applications.
- 1 to 6 blades to run the CEP with STREAM-OPS continuous queries. Each CEP node is configured with up to 6GB of memory.
- 1 blade to run the Ganglia monitor Server

We executed several experiments varying from 1 to 6 the number of nodes used to run the CEPs. In particular we used the 4 setups highlighted in Figure 3 with the CEP deployed in 1, 2, 4 and 6 nodes. The STREAM-OPS queries are parallelized in order to be deployed in the distributed CEP cluster.

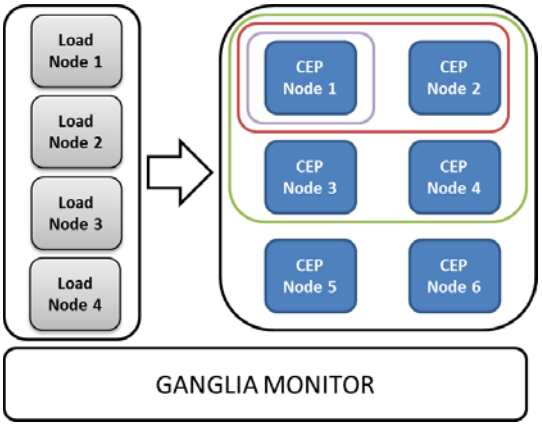

Figure 3: Deployments

3.2 Results

Figure 4 shows the maximum throughput reached by LeanBigData(LBD) and CoherentPaaS (CP) CEPs using the STREAM-OPS library in the 4 deployments.

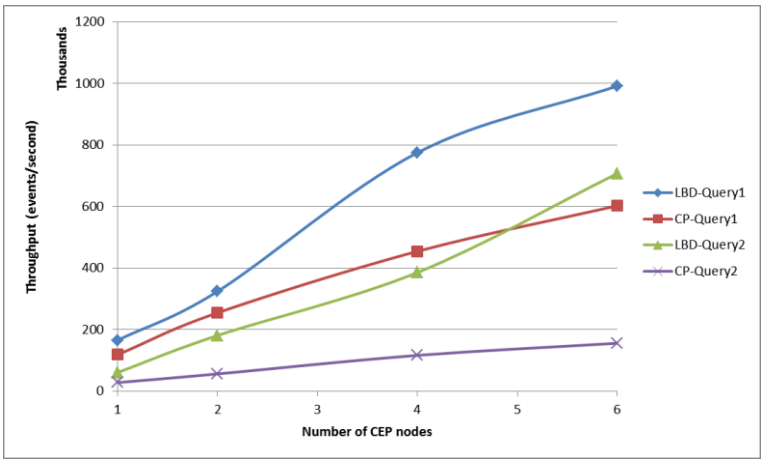

Figure 4: CEP Throughput comparison

In both scenarios the CEPs scale almost linearly reaching the million tuples per second processed with *query1* in the 6 node deployment of LBD-CEP. The difference in throughput between the two CEPs is due to very high efficient architecture of LeanBigData CEP that is able to optimize the distributed event processing. The STREM-OPS operators have the same behavior in the two CEPs. This is highlighted with Figure 5 and Figure 6 that report the latency of aggregate operator in the two

CEPs. The figures show the latency of one aggregate operator instance of query1 when used in the 4 node deployment. In both cases the average latency of the operation is around 15 milliseconds even if the throughput of LBD-CEP doubles the CP-CEP one.

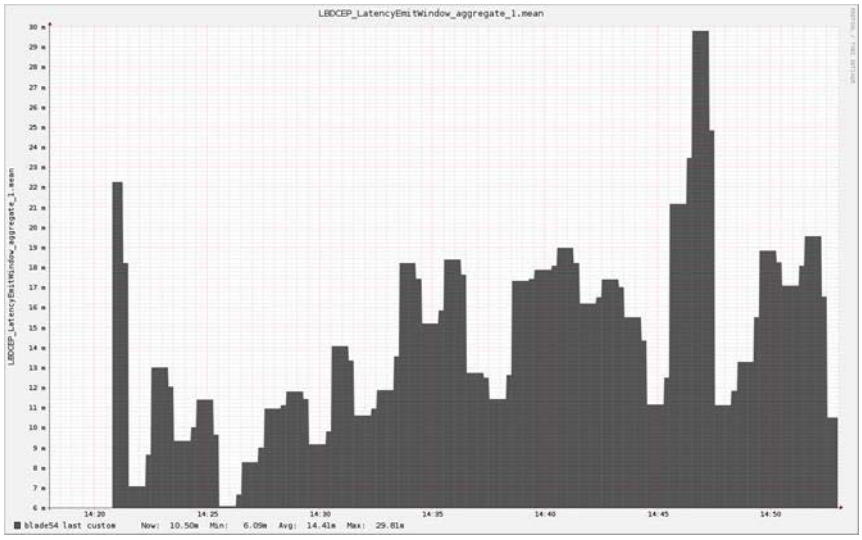

Figure 5: Aggregate Latency in LBD-Query1 with 4 nodes deployment

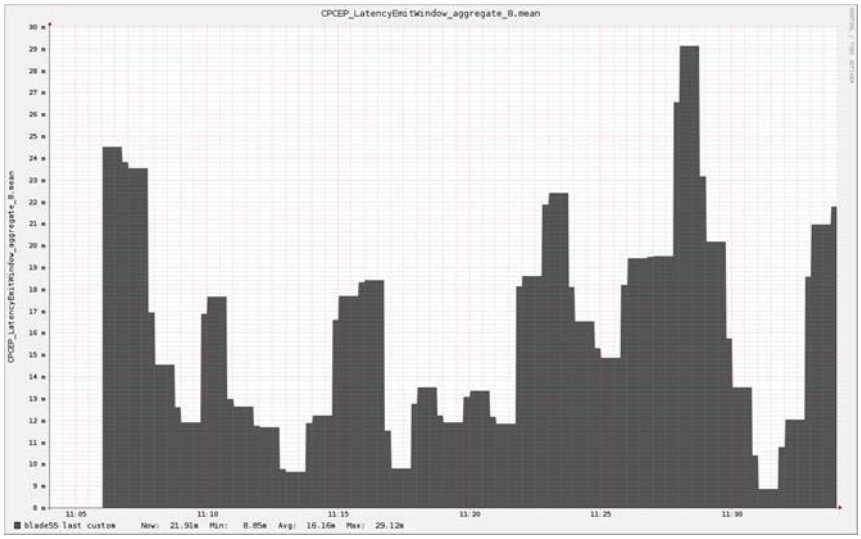

Figure 6: Aggregate Latency in CP-Query1 with 4 nodes deployment

4 Conclusions

In this paper we presented STREAM-OPS, a library of streaming operators written in JAVA that is designed to ease the process of streaming query definition. We demonstrate the efficiency of the library deploying it in two different Complex Event Processing engines able to process millions of events per second using few nodes with commodity hardware.

For the upcoming work, we plan to extend the library incorporating new streaming operators for text processing and to evaluate the library also on top of Apache Flink [5].

Acknowledgements

This research has been partially funded by the European Commission under projects CoherentPaaS and LeanBigData (grants FP7-611068, FP7- 619606), the Madrid Regional Council, FSE and FEDER, project Cloud4BigData (grant S2013TIC-2894), and the Spanish Research Agency MICIN project BigDataPaaS (grant TIN2013-46883)

References

1. Storm, Apache Storm web page. http://storm.apache.org/ Last visited 20/04/2016
2. CoherentPaaS. CoherentPaaS project web site http://www.coherentpaas.eu Last visited 20/04/2016
3. LeanBigData. LeanBigData project web site http://leanbigdata.eu/ Last visited 20/04/2016.
4. Borealis. The Borealis project, http://cs.brown.edu/research/borealis/public/ Last visited 20/04/2016
5. Flink. Apache Flink web page, https://flink.apache.org/ Last visited 20/04/2016
6. Spark. Apache Spark streaming web page, http://spark.apache.org/streaming/ Last visited 20/04/2016
7. Ganglia Web Page. http://ganglia.sourceforge.net Last visited 20/04/2016