UNIVERSITÉ LIBRE DE BRUXELLES ULB

Stream Processing with StreamSQL, Apache Kafka

Kshitij Kumar 000456998
Quang Duy Tran

INFO-H-415 Advanced Databases
Project Report

December 2017

Contents

List of Figures

1 Introduction

1.1 Streaming data

Data being generated continuously over time, from a large number of different data sources at high speed is considered as streaming data. The term is usually used in the context of big data. The important characteristics associated with streams, which also motivate the development of a different type of processing system, are:

- **Unboundedness** – the most important factor which characterises streaming data; it is continuously generated.
- **High velocity** – typically, it tends to be generated at a high rate rendering it infeasible to store and work with main memory
- **Low latency** - usually due to the unboundedness and high velocity, there spawns a need for fast processing: real time or near real time

The Figure 1 illustrates a basic schematic for a Stream Data Management System.

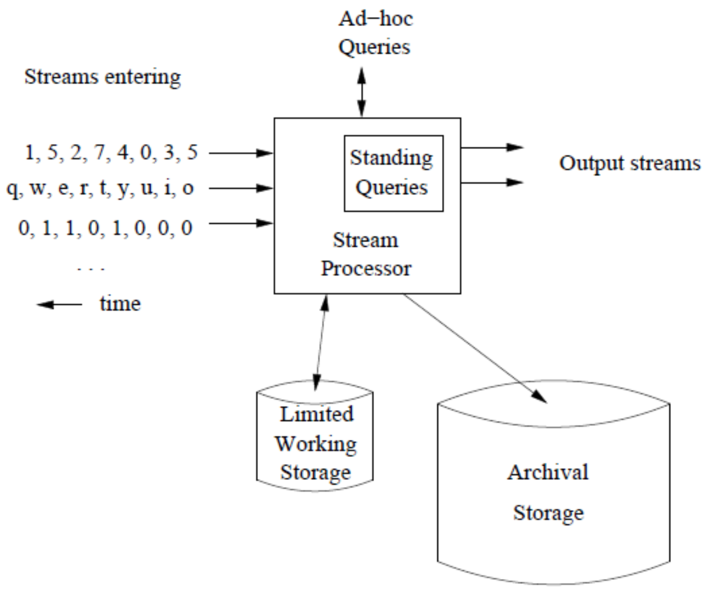


Figure 1: Schematic diagram of a Stream Processing System

Streaming data and real time applications based on it are plentiful. Some stream data sources are:

- Internet of Things, Sensors: Smart Cities, Driverless cars
- Internet, web traffic – clickstream
- Images – Satellite, CCTV
- Finance – Stocks, Transactions

- Factory manufacturing processes
- Logs - generated by automated systems

Some of the typical use cases include:

- Fraud detection based on financial transactions
- Investment decision based on stock prices
- Personalisation of an application, website etc based on user preference in real time
- Real time dashboards and monitoring, eg road traffic monitoring and identification of traffic rule violations
- Monitoring of a manufacturing process in a factory

1.2 Stream processing and Batch processing

Traditional, bounded data is served well by batch processing systems. Since, the data is known and finite, the whole data is fetched before performing any processing (Figure 2) It may take hours or even days to finish the query and deliver the results. In order to handle streams, with low latency being an important criterion, there is a need for optimisation of existing processing systems to handle this scenario.

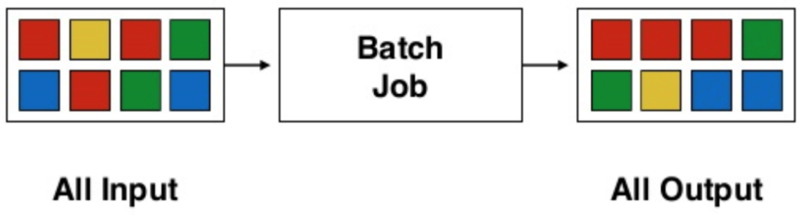


Figure 2: Batch processing

In contrast, stream processing reads data as it is generated over time and then processes continuously. The faster and earlier, the data is processed, the more valuable is the data. Typically, it is analyzed in the memory before written to disk, or even not written at all because the data is unpredictable and infinite.
(Figure 2)

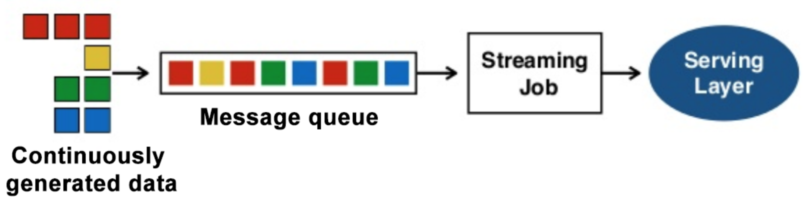


Figure 3: Stream processing

The need for stream processing systems have increased in the recent past mainly because of the availability of multiple sources of streaming data and the perceived opportunity to utilise this and arrive at faster decision making by lowering the latency.

1.3 Lambda Architecture

Historically, stream processing has been associated with incorrect results. This can be attributed to the fact that approximation algorithms were used to perform the high velocity operations typically demanded by these applications.

In most present day big data architectures, a stream processing system is used alongwith a batch processing system. The idea is to combine the best offered by both worlds. Systems can utilise the low latency, albeit at a slight loss to inaccuracy. The batch system comes in after some time, providing the correct results. This kind of system is usually realised with the help of Lambda Architecture. The Figure 4 gives a simple layout for this.

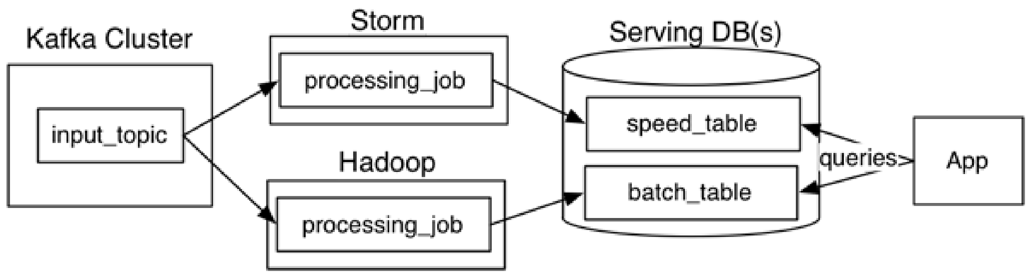


Figure 4: Simple layout of Lambda Architecture

Typically, the system consists of a data source, in this example it is a kafka cluster serving the messages. A stream processing engine and a batch processing engine then simultaneously ingest this data and basically perform the same calculation. In the framework of an Apache system, this typically is Apache Storm and Apache Hadoop respectively. The results of these are written to the application database when available and merged subsequently. The application can then query this database. This kind of systems are prevalent with many advantages. But, as is evident, the maintenance is often tricky and requires the building and provisioning of two independent pipelines and a final merge.

Lately, the stream processing offerings have been getting better in terms of providing higher accuracy. It is definitely possible to use, and there are reportedly many systems, which are composed entirely of stream processing pipelines. Some proponents even make the case of completely letting go of batch systems for most use cases. Their argument being that the stream processing systems are already beginning to match the performance on other parameters, in addition to lower latency. [2][3]

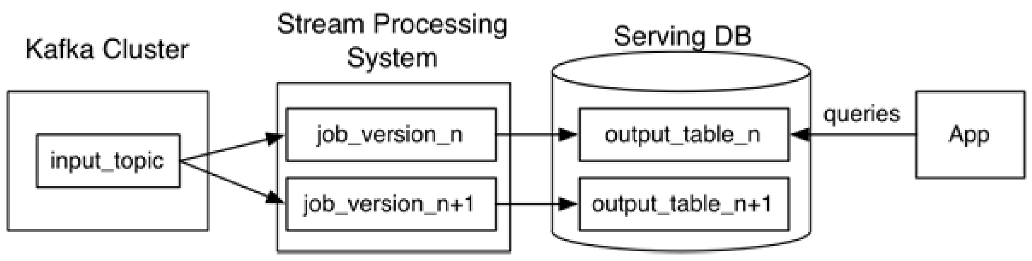


Figure 5: A stream processing based architecture

1.4 Landscape

Around the beginning of this millennium, as the requirements of handling stream data was being realised, many systems were being developed within academica. Some examples include Stanford University's STREAM (Stanford Stream Data Manager) [4], Aurora[5], Borealis[6], Medusa[7] among others. There is not much active development on these systems anymore. The Figure 6 shows a schematic of the Medusa system, from their homepage[7].

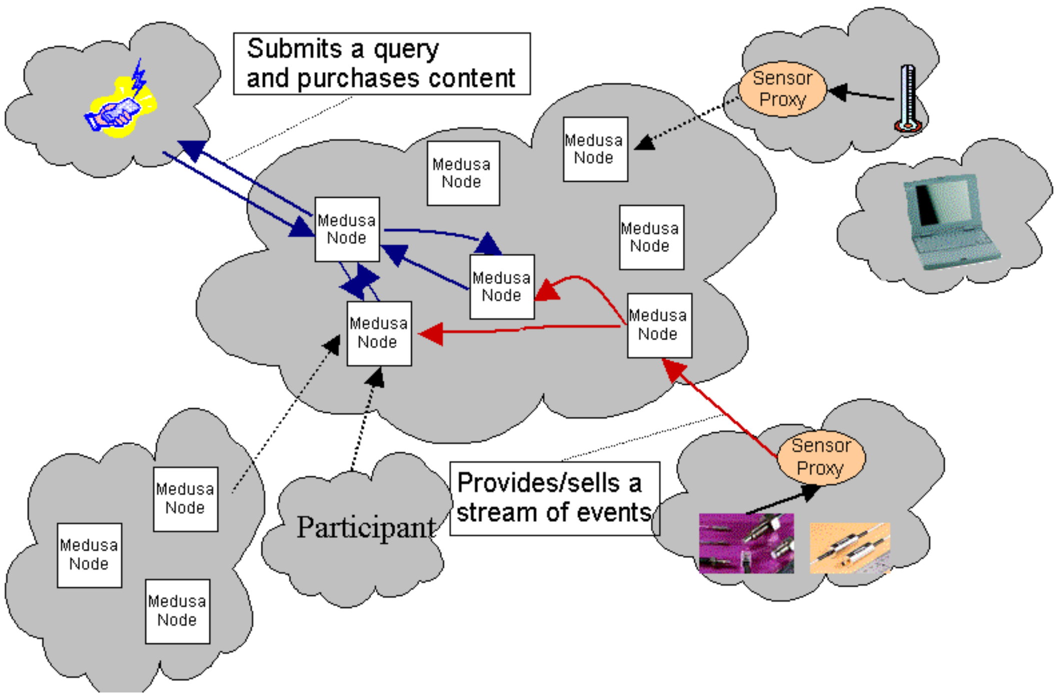

Submits a query and purchases content
Sensor Proxy
Medusa Node
Participant
Provides/sells a stream of events
Sensor Proxy

Figure 6: Medusa architecture

Subsequently, there have been various other independent systems also developed in the industry which typically build on the ANSI SQL standard to provide streaming functionality. PipelineDB[8] and StreamSQL[9] are some products available. StreamSQL is discussed in greater detail in the later sections of the report.

Recently, with the emergence of large amounts of data, many new tools are being developed in the industrial space. Moreover, many of these have been open sourced or in some cases, their designs have been made available on the web. The Apache ecosystem has its fair share of offerings: Apache Storm, Apache Spark Streaming, Apache Flink, Apache Samza and others. All the above are distributed real-time processing engines and typically use Apache Kafka within the pipeline for message communication.

All the above are meant for the same purpose of real time computations, but the inherent distinctions in design choices make for an interesting variety in terms of selection for an application. Apache Storm is oriented specifically for streams and the most popular system in use. Apache Spark is inherently a batch processing engine and its Streaming extension models streams by using mini batches. Interestingly, Apache Flink is based with design considerations for stream processing, but it also provides batch processing capabilities which are modeled on top of the stream ones. Additionally, around August 2017, Apache Kafka also launched a developer's preview of KSQL which is based on its own StreamsAPI. This also provides the possibility to construct real time analytical queries directly on Kafka topics.

Cloud based services are also available which remove the need for managing the immense amount of hardware required for maximizing the performance obtained from these distributed processing systems. The most prominent ones are Google Data Cloud, Amazon Kinesis and Microsoft Azure Streaming Services.

1.5 Stream Data Model Considerations

The basic considerations in a stream data model are:

1. Windowing: The unboundedness and the velocity of data arriving at the processing engine makes it infeasible to store everything in active storage. Hence, a sliding window of the most recently arrived data is used for computations and sometimes, the summaries are also kept for some kind of ad-hoc queries.

2. Time: It is a real time system and latency is arguably the most important factor. Hence, there is a notion of event time and processing time. Event time, as the name suggests, corresponds to the actual time of the occurence of the event under consideration. Whereas, the processing time is the time of arrival of the event at the processing engine.

2 SQLStream

SQLStream is an enterprise software which serves as a real-time platform and provides SQL-compliant stream analytics for both developers and analysts.

2.1 Components

It has the following components:

- **SQLstream s-Server** : a fully compliant, distributed, scalable and optimized SQL query engine for unstructured data streams.
- **SQLLineClient** : a command line interface for s-Server
- **StreamLab** : a visual platform for interactively exploring data streams and building applications in a web based graphical user interface
- **s-Dashboard** : an HTML5 platform for building push-based, real-time visualizations on s-Server
- **s-Studio**: an integrated development environment for stream exploration, application development, and administration for the s-Server platform. It enables dynamic updates to live applications, adding new queries as needed or changing existing queries or views.

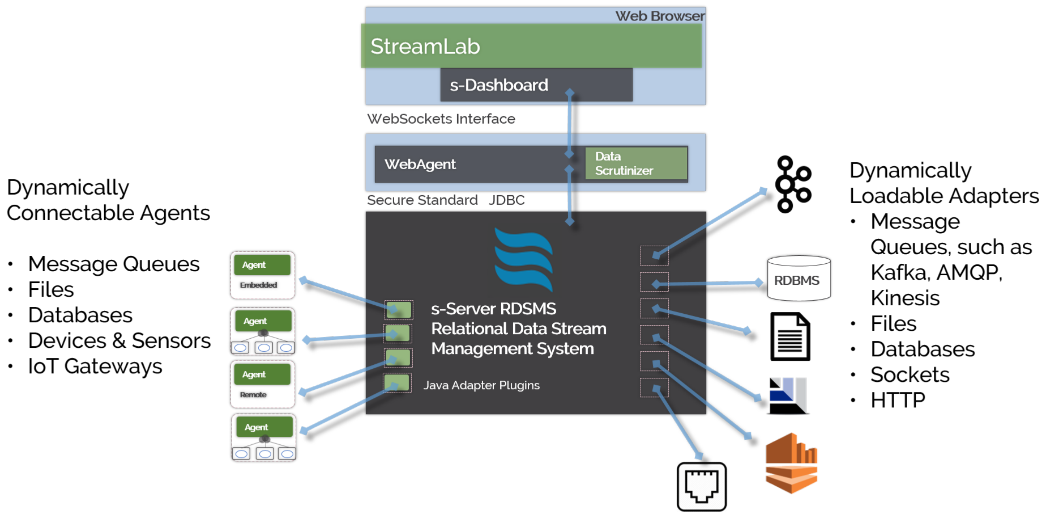


Figure 7: SQLStream overview

2.2 Pipeline

The data pipelines within a SQLstream application is quite straightforward. There is an input data source, which provides the stream data flowing in at one end, which is subsequently analyzed based on application specific logic, with the final computation results as an output at the other end. The analysis is typically carried out with custom views which act as continuous queries. There is a SQLStream specific construct, a pump, which essentially encapsulates the selection and insertion of data in the application context. The Figure 8 provides a rough schematic of a typical pipeline.

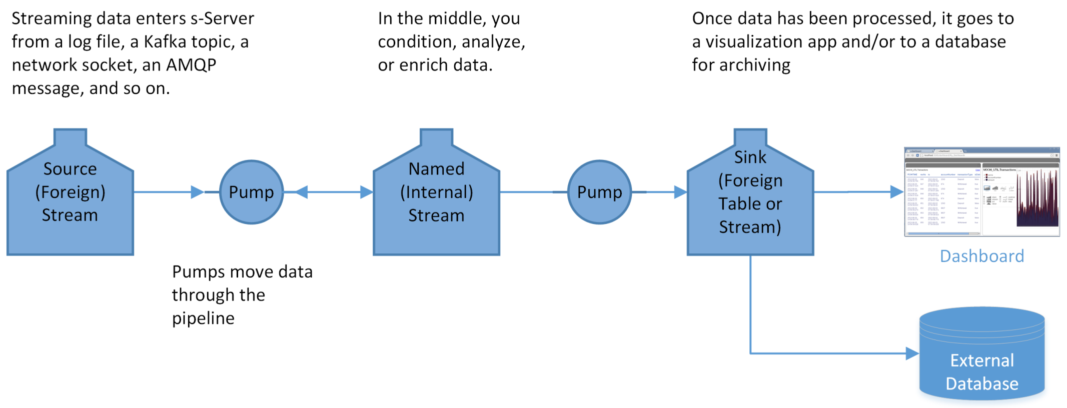

Streaming data enters s-Server from a log file, a Kafka topic, a network socket, an AMQP message, and so on.
In the middle, you condition, analyze, or enrich data.
Once data has been processed, it goes to a visualization app and/or to a database for archiving
Source (Foreign) Stream
Pump
Named (Internal) Stream
Pump
Sink (Foreign Table or Stream)
Dashboard
Pumps move data through the pipeline
External Database

Figure 8: A typical SQLStream pipeline

2.3 Streaming Data considerations within SQLStream

The pipeline is processed as a continuous flow. A streaming query is a continuous, standing query that executes over streaming data and is implemented as views. It is similar to traditional relational database views, with the important distinction of being continuously updated as and when the stream comes in.

Moreover, some queries utilise operations which require finite data available to them to be able to calculate the result, for example, aggregates and joins. These are typically handled by the concept of windowing. These can be in two forms. One based on a fixed number of rows and another based on time, eg rolling or periodic time. The rolling time considers continuous, joint subsets of time, eg the last second, the last five seconds and so on. Periodic time considers disjointed regular intervals of time, eg an hourly report.

Streaming data is time-sensitive data and is represented as sequences of time-stamped messages, which is called ROWTIME in the context of SQLStream.

2.4 SQLStream objects

The main objects are described below:

- **Stream**: A stream is the primary building block in the whole system. It implements a publish-subscribe protocol and can be written and read by multiple agents.

- **Foreign Stream**: A stream defined in the context of a schema and associated with an external system.

- **Foreign Table**: An object that records the metadata necessary for SQLstream sServer to be able to access a table or a similar data structure in a remote database

- **View**: A relation that provides a reusable definition of a query and acts as continuous queries.

- **Pump**: A SQLstream schema object providing a continuous query in the form of a sequenced INSERT INTO followed by a SELECT FROM query functionality. It basically selects data from one source, a stream or table, and inserts it into a sink, another stream or table.

- **Objects for communicating with external applications**: Foreign Data Wrappers, Server Objects and User-Defined Routines

2.5 Querying operations

The querying operations are implemented on the existing framework provided by the ANSI SQL standard. Most of the definitions and semantics still hold with some additional constructs.

In a streaming context, due to the unboundedness of streams, querying statements like SELECT essentially run forever. It is a blocking call which runs until the next row of the stream becomes available or the statement is explicitly closed.

It is also possible to connect to tables from external databases and perform combined operations like joins, aggregates etc with multiple streams and tables.

Additionally, operations like aggregates and joins require a finite set of records and must have all the data to produce the correct results. This is handled by windowing, as mentioned in a previous section. A specific keyword, WINDOW, is used to define the window parameters: type and size.

All the basic building blocks, including SQLstream Data Types and Streaming SQL Operators are similar to SQL. The standard SQL operators like CREATE statements, DROP statements, SELECT statements, INSERT, MERGE, DELETE, ALTER statements, arithmetic and logical operators are all available. The operators for transforming and filtering incoming data is updated and consists of WHERE, JOIN, GROUP BY and WINDOW.

Connecting to external applications is possible through JDBC, which makes SQLstream s-Server look like a database. External applications can then send queries to streams and obtain results received from a stream.

2.6 Building Applications

A typical application flow is detailed in Figure 9. In the figure, the primary development environment is StreamLab, the web based graphical user interface.

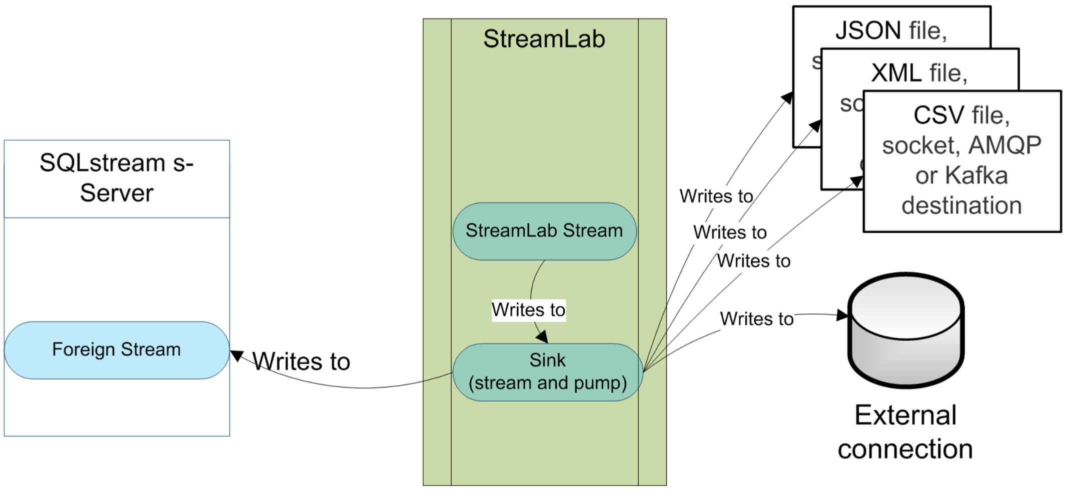


Figure 9: Schematic of a SQLStream application

The pipeline begins with the creation of a source. The different options can be selected from the menu, by dragging and dropping it into the sources frame. The parameters for the data source can be modified here. Figure 10 shows this screen.

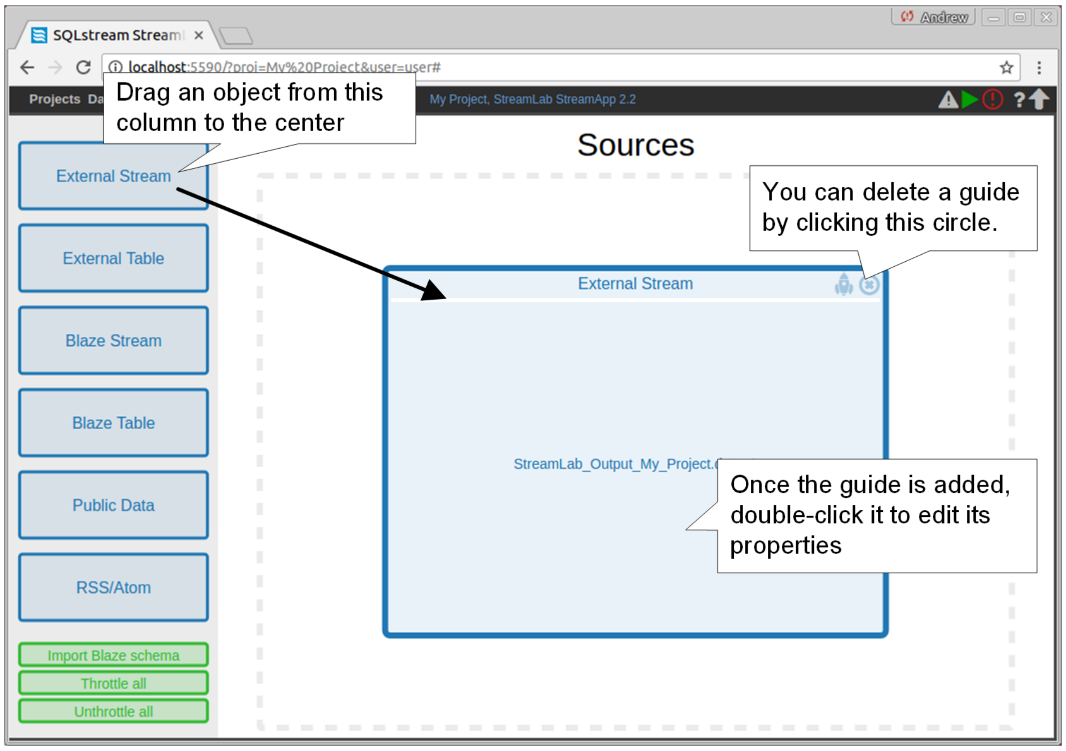
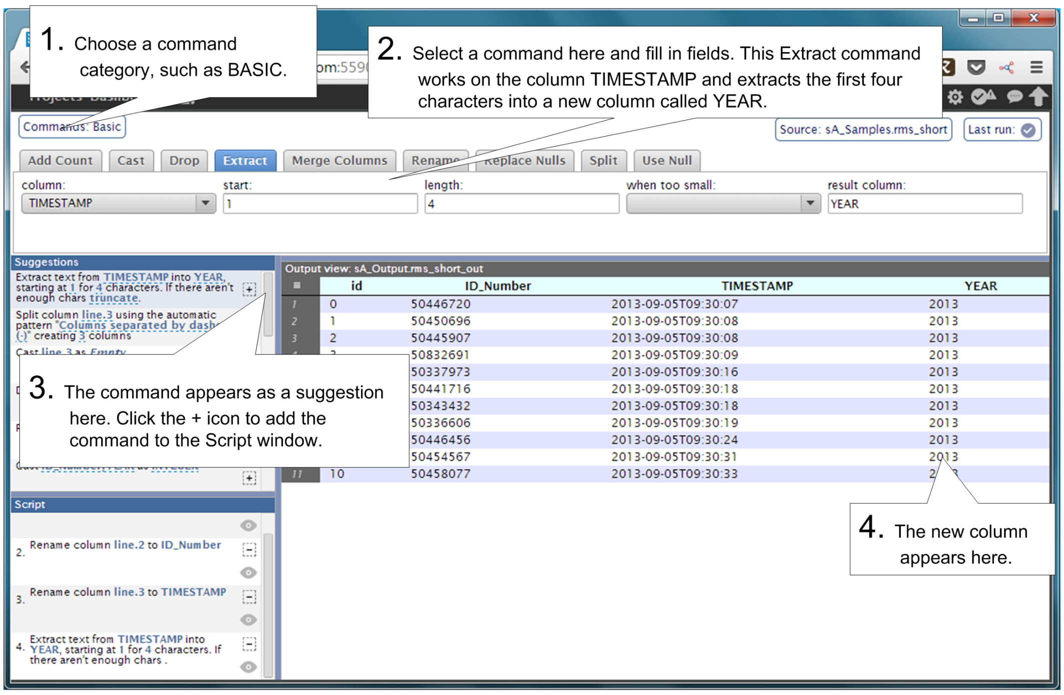

SQLstream Stream
localhost:5590/?proj=My%20Project&user=user#
Projects Da
My Project, StreamLab StreamApp 2.2
Drag an object from this column to the center
Sources
External Stream
External Table
Blaze Stream
Blaze Table
Public Data
RSS/Atom
Import Blaze schema
Throttle all
Unthrottle all
You can delete a guide by clicking this circle.
External Stream
StreamLab_Output_My_Project.
Once the guide is added, double-click it to edit its properties

Figure 10: Adding a data source in StreamLab

Once the source is selected, the next analysis steps are constructed as guides, in SQLStream terminology, which are basically views that transform the data. There are many different pre-built functions, called as commands, available here. A screenshot of a StreamLab window showing some of the components and operations that can be performed is shown in Figure 11.


1. Choose a command category, such as BASIC.
2. Select a command here and fill in fields. This Extract command works on the column TIMESTAMP and extracts the first four characters into a new column called YEAR.
3. The command appears as a suggestion here. Click the + icon to add the command to the Script window.
4. The new column appears here.
Commands: Basic
Source: sA_Samples.rms_short
Last run:
Add Count
Cast
Drop
Extract
Merge Columns
Rename
Replace Nulls
Split
Use Null

column: TIMESTAMP
start: 1
length: 4
when too small:
result column: YEAR

Suggestions

Extract text from TIMESTAMP into YEAR, starting at 1 for 4 characters. If there aren't enough chars truncate.

Split column line.3 using the automatic pattern "Columns separated by dash (-)" creating 3 columns

Script

2. Rename column line.2 to ID_Number

3. Rename column line.3 to TIMESTAMP

4. Extract text from TIMESTAMP into YEAR, starting at 1 for 4 characters. If there aren't enough chars .

Output view: sA_Output.rms_short_out

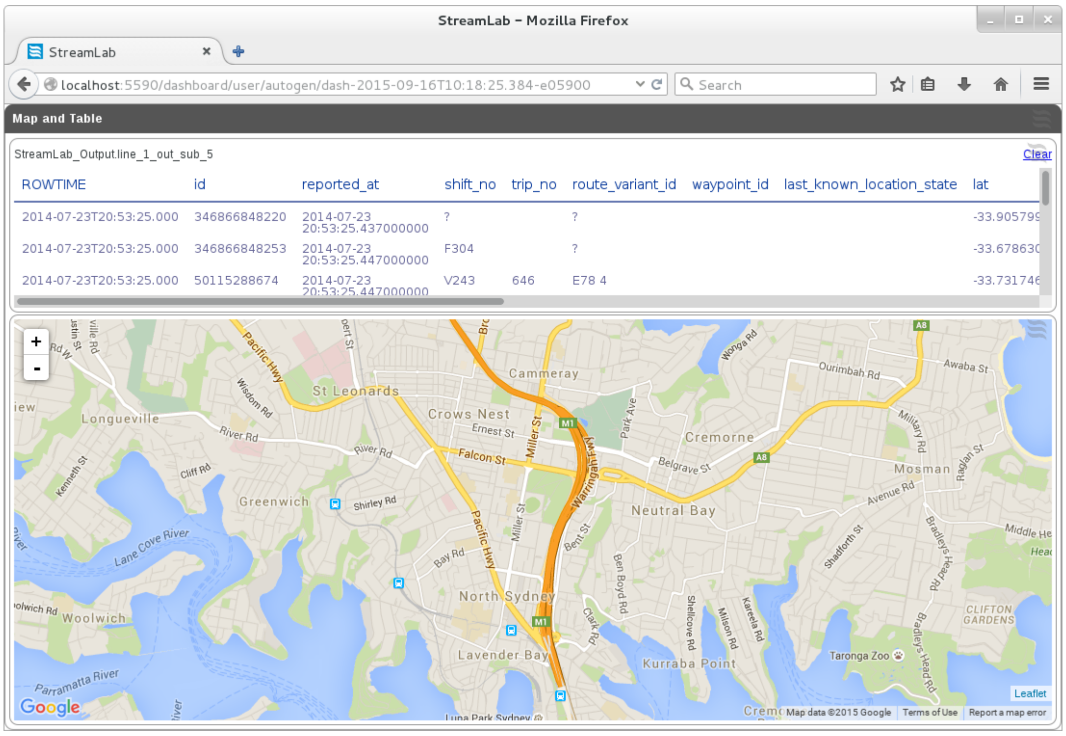
	id	ID_Number	TIMESTAMP	YEAR
1	0	50446720	2013-09-05T09:30:07	2013
2	1	50450696	2013-09-05T09:30:08	2013
3	2	50445907	2013-09-05T09:30:08	2013
		50832691	2013-09-05T09:30:09	2013
		50337973	2013-09-05T09:30:16	2013
		50441716	2013-09-05T09:30:18	2013
		50343432	2013-09-05T09:30:18	2013
		50336606	2013-09-05T09:30:19	2013
		50446456	2013-09-05T09:30:24	2013
		50454567	2013-09-05T09:30:31	2013
11	10	50458077	2013-09-05T09:30:33	2

Figure 11: Stream Analysis functions

The possible commands are listed below:

- **Basic**: Rename, split, merge etc. These are also shown in the Figure 11 top row.
- **Aggregate** a particular column, including COUNT, SUM, and AVERAGE.
- **Calculate** arithmetical calculations on a numerical column
- **Categorize** a continuous value by applying conditions to the column.
- **Running Average** creates a new column that is a running average of another column over a given window of time.
- **Time Sort** uses a sliding time-based window of incoming rows to sort by the selected column or by ROWTIME
- **Window** To perform a windowed aggregation on a selected column.
- **GroupRank** To group rows by the values in one column (group by) and then rank the rows within that group according to another column (rank by) across a window
- **Partition Window** To partition rows into groups using the columns in a particular column.
- **Table Lookup** To enrich available data with external data from databases

Once, the calculations are performed, it is possible to store these to an external database or other files. SQLStream has an inbuilt component, dashboard, for visualization of the data. There are many prebuilt visualizations available based on time series graphs. There is also the possibility to plot geospatial data on maps. An example of this is shown in Figures 12 and 13.


StreamLab – Mozilla Firefox
StreamLab
localhost:5590/dashboard/user/autogen/dash-2015-09-16T10:18:25.384-e05900
Search
Map and Table
Clear
Leaflet
Map data ©2015 Google
Terms of Use
Report a map error

StreamLab_Output.line_1_out_sub_5

ROWTIME	id	reported_at	shift_no	trip_no	route_variant_id	waypoint_id	last_known_location_state	lat
2014-07-23T20:53:25.000	346866848220	2014-07-23 20:53:25.437000000	?		?			-33.905799
2014-07-23T20:53:25.000	346866848253	2014-07-23 20:53:25.447000000	F304		?			-33.678630
2014-07-23T20:53:25.000	50115288674	2014-07-23 20:53:25.447000000	V243	646	E78 4			-33.731746

Figure 12: Map and table based visualization in s-Dashboard

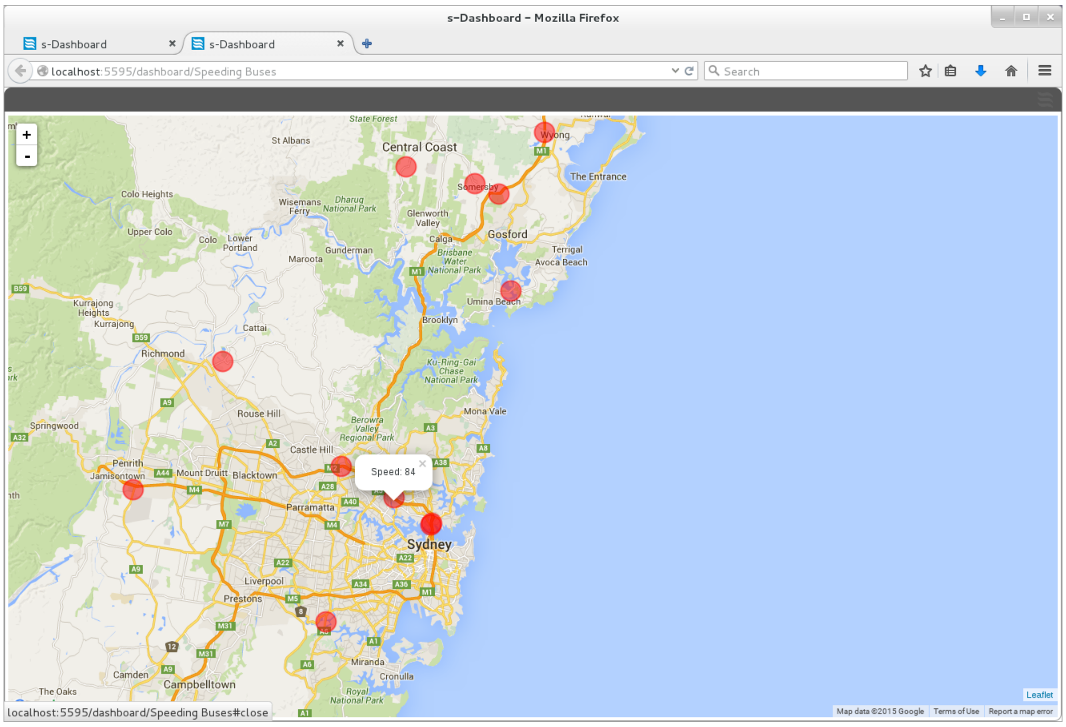


Figure 13: Map based visualization in s-Dashboard, featuring filtered records

2.7 SQLStream, Relational Databases

The inherent data model of SQLStream is shared with and is fundamentally a common data model centered on processing relation oriented constructs, eg rows, queries, and views. The querying constructs are also shared: data manipulation and definition languages standardized as SQL.

The differences which exist are primarily to account for the unboundedness of streams. s-Server uses predetermined queries over streaming data, with continuous processing and continuous querying. On the other hand, a Relational Database is used for ad-hoc queries over batch data, with querying performed on this fixed set of data. The queries in streaming scenario are usually scoped over explicit time windows based on the application requirements.

Moreover, the two can work well together and can be seamlessly integrated. As an example, s-Server can use static predetermined queries to process data for a Relational Database and also respond to incoming messages by triggering dynamic queries on the stored data.

2.8 Application Properties

SQLStream's philosophy is to implement an end-to-end system within a stream management system, which communicates to an application for computation intensive and complex calculations. So, basically, SQL has control and the programming language is embedded within the application.

3 Apache Kafka, KSQL

Apache Kafka is a stream processing platform based on a publisher-subscriber message queue architecture model. It is designed for distributed deployment and has a Java based library, Kafka streams, for stream processing. There is also the option to connect to external systems. Kafka organises messages in topics, with multiple producers and consumers.

Recently, during the end of August, 2017, KSQL was launched as a developer preview. KSQL is a streaming SQL engine which internally uses Kafka streams. But, with KSQL, it is now possible to directly query the data without writing external applications. There are various stream based operators available, a subset of which includes aggregations, joins, windowing, sessionization.

Most of the semantics is similar to the ones discussed above for the SQLStream system. From the roadmap, it seems that more and important capabilities are being added to the next version of KSQL.

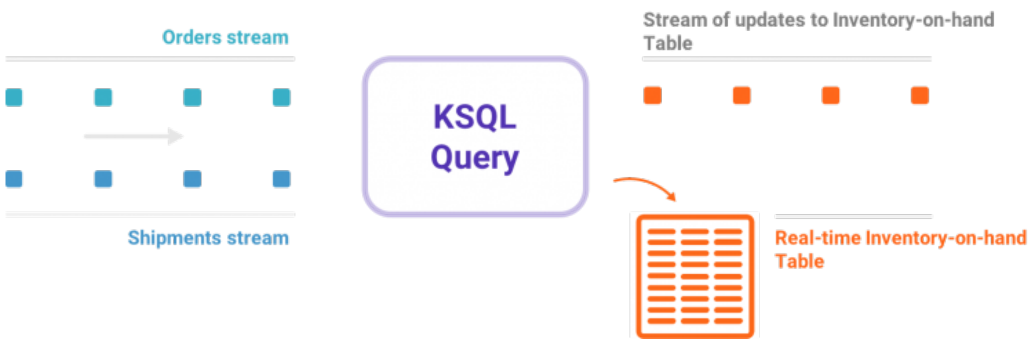

Orders stream
Shipments stream
KSQL Query
Stream of updates to Inventory-on-hand Table
Real-time Inventory-on-hand Table

Figure 14: A streaming analysis scenario with KSQL

4 Application

Some application scenarios were considered to get a first experience in using and understanding the stream processing systems.

4.1 Clickstream Analysis

The proliferation of websites and the need to generate revenue from the online presence has led to the emergence of clickstream analytics. A clickstream is fundamentally a stream based record of clicks on a website. This kind of website activity is logged and gaining insights from this data is lucrative. It is also possible to collect this data from user browsers.

A clickstream typically contains information about a page request based on the protocol, eg HTTP GET, HTTP 404 error etc, ip information of the user, web-agent of the user, time of the event and possibly additional information. In our analysis, we use the following schema:

```
clickstream (
_time bigint,
time varchar,
ip varchar,
request varchar,
status int,
userid int,
bytes bigint,
agent varchar)
```

A brief explanation of the attributes of the stream:

- _time and time correspond to the time of the click. One is a numeric attribute, storing the milliseconds since January 1, 1970 and the other is storing time in a varchar format which corresponds to datetime
- ip is the user ip address
- request, status are the protocol request and status packet information
- userid is the user id assigned to the users stored in the system. These are typically available for registered users on the website
- bytes is the size of the packet used for the information communication, eg if it is a simple HTTP GET Request, then it will just contain the size of the text containing the packet information as required by the communication protocol. If a file is being transmitted, the size of the file is also part of the message.
- agent is the web agent, typically a browser which is being used to access the website.

This data is generated within kafka. A kafka and ksql installation has some data generator topics already created and that was used to generate the data for the analysis.

Two other tables, users and status_codes are also created, within separate kafka topics. Their descriptions are as follows:

```
users (
user_id int,
registered_At long,
username varchar,
first_name varchar,
last_name varchar,
city varchar,
level varchar)

status_codes (
code int,
definition varchar)
```

The aforementioned three data sources, one streaming and the other two tables, in the form of kafka topics were added as data sources for the SQL-Stream s-server. For KSQL, the possible data sources are only kafka topics, which are provided during the creation of the corresponding stream or table.

The initial setup for s-server looks as follows:

```
--To access the external kafka topic, a Server object needs to be created
which encodes the connection parameters
CREATE OR REPLACE SERVER "KafkaServer" TYPE 'KAFKA'
FOREIGN DATA WRAPPER ECDA;

--This is the namespace of the application
CREATE OR REPLACE SCHEMA "clickstream";
SET SCHEMA '"clickstream"';

--The following creates the stream
CREATE OR REPLACE FOREIGN STREAM "clickstream_source"
("_time" bigint NOT NULL,
"time" varchar NOT NULL,
"ip" varchar NOT NULL,
"request" varchar NOT NULL,
"status" int NOT NULL,
"userid" int NOT NULL,
"bytes" bigint NOT NULL,
"agent" varchar NOT NULL)
SERVER "KafkaServer"
OPTIONS
(topic 'clickstream',
seed_brokers 'localhost',
starting_time 'latest',
parser 'JSON',
character_encoding 'UTF-8',
skip_header 'false');

--Similarly, for the  other two topics

users (
"user_id" int NOT NULL,
"registered_At" long NOT NULL,
"username" varchar NOT NULL,
"first_name" varchar NOT NULL,
"last_name" varchar NOT NULL,
"city" varchar NOT NULL,
"level" varchar NOT NULL)
SERVER "KafkaServer"
OPTIONS
(topic 'users',
seed_brokers 'localhost',
starting_time 'latest',
parser 'JSON',
character_encoding 'UTF-8',
skip_header 'false');

status_codes (
"code" int not null,
"definition" varchar not null)
SERVER "KafkaServer"
OPTIONS
(topic 'status_codes',
seed_brokers 'localhost',
starting_time 'latest',
parser 'JSON',
character_encoding 'UTF-8',
skip_header 'false');
```

The initial setup for ksql is below:

```
CREATE STREAM clickstream (
_time bigint,
time varchar,
ip varchar,
request varchar,
status int,
userid int,
bytes bigint,
agent varchar)
with (
kafka_topic = 'clickstream',
value_format = 'json');

CREATE TABLE users(
user_id int,
registered_At long,
username varchar,
first_name varchar,
last_name varchar,
city varchar,
level varchar)
with (
key='user_id',
kafka_topic = 'users',
value_format = 'json');

CREATE TABLE status_codes (
code int,
definition varchar)
with (
key='code',
kafka_topic = 'status_codes',
value_format = 'json');
```

Some of the queries which are tried:

- Monitoring user activity: This basically requires the creation of a continuous query which joins the two data sources, clickstream and users, subsequently performing a selection for a particular user over the time span of last few minutes.

  ```
  --SQLStream s-server
  --Note: A WINDOW can be time based(RANGE keyword) or row based(ROWS keyword)
  CREATE OR REPLACE STREAM user_clickstream AS
  SELECT STREAM userid, u.username, ip, u.city, request, status, bytes
  FROM clickstream OVER (RANGE INTERVAL '3' SECOND PRECEDING) AS c
  LEFT JOIN users u ON c.userid = u.user_id;

  CREATE VIEW user_activity
  AS SELECT STREAM username, ip, city, COUNT(*) AS count
  FROM user_clickstream
  GROUP BY username, ip, city HAVING COUNT(*) > 1;

  --ksql
  CREATE STREAM user_clickstream AS
  SELECT userid, u.username, ip, u.city, request, status, bytes
  FROM clickstream c LEFT JOIN users u ON c.userid = u.user_id;

  CREATE TABLE user_activity
  AS SELECT username, ip, city, COUNT(*) AS count
  ```

```
FROM user_clickstream
WINDOW TUMBLING (size 180 second)
GROUP BY username, ip, city HAVING COUNT(*) > 1;
```

- Some aggregation based queries to get a summary of the maximum, minimum and average number of requests and error over the last time window, eg 3 minutes.
- Reconstruction of user sessions was implemented. A session is assumed to be over when there is 5 minutes of inactivity for a particular user.

4.2 Taxi Trips

The dataset was used in Distributed Event-Based Systems (DEBS) 2015 grand challenge and is based on an open data source consisting of taxi trip information in New York City, USA. [11] For our analysis, we used the smaller dataset corresponding to the first 20 days of the data, with 2 million events and about 375MB extracted data size of the csv file. The full dataset consists of records for the whole of 2013 and 173 million events.

The schema and description of the attributes of the data set are as follows:

```
taxi_trip(
--an md5sum of the identifier of the taxi vehicle bound
medallion varchar,
--an md5sum of the identifier for the taxi license
hack_license varchar,
--time when the passenger(s) were picked up
pickup_datetime
--time when the passenger(s) were dropped off
dropoff_datetime
--duration of the trip
trip_time_in_secs
--trip distance in miles
trip_distance
--longitude coordinate of the pickup location
pickup_longitude
--latitude coordinate of the pickup location
pickup_latitude
--longitude coordinate of the drop-off location
dropoff_longitude
--latitude coordinate of the drop-off location
dropoff_latitude
--the payment method: credit card or cash
payment_type
--fare amount in dollars
fare_amount
--surcharge in dollars
surcharge
--tax in dollars
mta_tax
--tip in dollars
tip_amount
--bridge and tunnel tolls in dollars
tolls_amount
--total paid amount in dollars
total_amount)
```

Since, this data is a regular dataset, it was required to be converted to a form of streaming dataset for analysis. This was done by simulating a log file. An initially empty log file was created and the sorted csv file was read from the beginning. A fixed number of lines was copied from the csv file to the log file. This number basically categorises the event rate of the stream and we varied this from 50, 100, 250, 500 and 1000.

The main aim of this analysis was to get an experience with the web based graphical interface StreamLab and the dashboard provided by StreamLab. Hence, everything was modeled as a graphical data flow.

The source was selected as a log file with a csv parser. Some simple calculation based queries were performed to obtain an idea of the events in a previous time frame, eg 30 minutes, 1 hour etc. Specifically, the longest and shortest trips based on the trip_time_in_secs and trip_distance, and also the cheapest and the most expensive trips were observed.

5 Some performance comparisons

SQLStream is an enterprise product and offers a trial license for a period of two months. The free version of the product has a limited usage and only 1GB data can be used per day. The pricing is $40k for base annual subscription, plus $10k per server and $1k per core. There is also a possibility to use it via Amazon Web Services on pay-as-you-use pricing model.

SQLStream, according to its website, is apparently the only SQL 2008 compliant stream processing engine. It has reported processing data at over 1M events/second per CPU core on benchmarks and at over 1M events/second per server on real-world applications.

Additionally, Amazon Kinesis Analytics is based in part on certain technology components licensed from SQLstream s-Server[10]

It provides an end-to-end system for stream data analysis. The presence of GUI based tools allow fast prototyping and also lends itself to be used by people not well versed with SQL. Inbuilt dashboards provides nice visualizations for exploration and data understanding. Also, the setup is quite simple, it has a couple of packaged installers which take care of all the installation procedures. Overall, the experience is quite good. One important thing to consider is that it's an enterprise software and quite expensive.

On the other hand, most big data systems modeled using Apache ecosystem components use Apache Kafka already. Hence, the availability of a sql based real time querying functionality directly on the kafka topics, provides interesting uses. The system is open-source which is an important consideration in terms of development of the project and easiness of adopting it. The syntax of ksql is concise and slightly easier to understand and write vis-a-vis for SQL-Stream s-server. Currently, it is still in developer preview and development is continuing to achieve more maturity be ready for deployment in production.

Latency is arguably the single most important factor for quantitatively comparing stream processing systems. Based on benchmarking done on some distributed system with streaming data, the latency ranges obtained for common systems are shown in Figure 15[12].

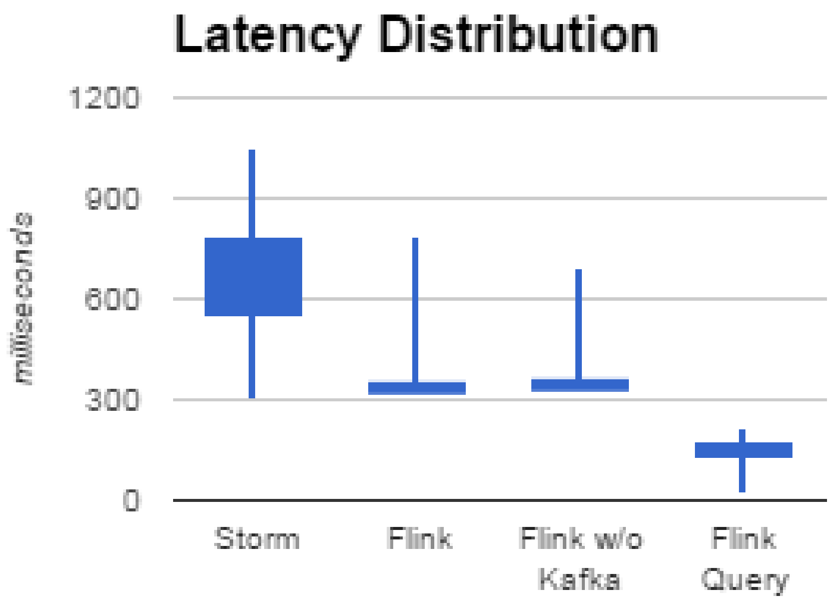


Figure 15: Latency measurements

Based on our analysis, the performance of both ksql and streamsql were comparable. Though extensive benchmarking was not pursued, on a single compute node with 16GB RAM and intel i7 7th generation processor, the latency of the aggregation queries based on the taxi trips data was well under 1s, for all event rates ranging from 50-1000 events per second.

6 Conclusion

Stream Processing is a broad topic with an immense scope. Within the scope of this project, however, the aim was to get a first overall view on stream processing in general and to test this through an application modeled in a suitable technology. SQLStream is a complete independent end-to-end package, with a stream processing engine, a command line interface for engine, a graphical user interface(both desktop and web-based) and visualization tools, making it amenable to get a first look without the potential problems of setting up and learning the different and ever changing landscape of technologies. To test it with more recent and open source developments, it was combined with Apache Kafka as the stream producer data source. Also, KSQL came out as a developer preview during the course of the project, hence, it was also tried with a simple application scenario. There are some things which unluckily could not be pursued in the present context. Specially important is the area of distributed processing, which is one of the cornerstones for the wide adoption of present day stream processing systems.

References

[1] Jure Leskovec, Anand Rajaraman, Jeff Ullman *Mining of Massive Datasets*. Cambridge University Press, United Kingdom, 2014.

[2] Tyler Akidau, *https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101* [Online]

[3] Jay Kreps, *https://www.oreilly.com/ideas/questioning-the-lambda-architecture* [Online]

[4] Stanford University STREAM system, *http://infolab.stanford.edu/stream/* [Online]

[5] Aurora system, *http://cs.brown.edu/research/aurora/*. [Online]

[6] Borealis system, *http://cs.brown.edu/research/borealis/public/* [Online]

[7] Medusa system, *http://nms.csail.mit.edu/projects/medusa/*. [Online]

[8] PipelineDB, *https://www.pipelinedb.com*, [Online]

[9] SQLStream, *http://sqlstream.com/*, [Online]

[10] SQLStream and Amazon Kinesis Analytics, *http://sqlstream.com/datasheet/cloud-sqlstreamamazon-kinesis-analytics/* [Online]

[11] DEBS 2015 Grand Challenge: Dataset and information, *http://debs.org/debs-2015-grand-challenge-taxi-trips/* [Online]

[12] Benchmarking of some Stream Processing Systems *https://data-artisans.com/blog/extending-the-yahoo-streaming-benchmark* [Online]

[13] A github repository providing links to various stream processing systems *https://github.com/manuzhang/awesome-streaming* [Online]