**Benchmark of windowed aggregations**

By: Patrick Lehmann

Mentor: Jeyhun Karimov

**Table of contents:**

1. Motivation/Goals
2. Project setup
   1. Latency
   2. Dataflow
   3. System setup
   4. Experiment setup
3. Results
   1. Experiment “workload”
   2. Experiment “window length”
   3. Experiment “sliding size”
4. Conclusion

# Motivation/Goals

Due to the various number of different streaming engines like Flink(), Spark(), Storm(), Apache Samza, Apache Apex, and Google Cloud Dataflow, it is from high interest to know how these systems behave in different situations. Especially as all have different characteristics and approaches in data processing and therefore their underlying architecture. For example, Spark uses microbatches to process data streams while other systems like Flink can process rows after rows of data in real time. Because many factors influence the performance this benchmark focuses on windowed aggregations. Therefore, it will analyze parameters like the window length, the sliding length, and the workload. To measure the performance the latency will be computed.

# Project setup

To generate the data stream a Taxi Ride file() was used in this benchmark. This contains of several taxi rides in the USA with parameters like starting time, number of passenger, and arriving time. This file was injected into Apache Kafka(), which produced the data stream that will be received by the streaming systems. This benchmark will compare the data streaming systems Flink and Spark. Both systems are used in practice by several companies and organizations. They also distinguish themselves through their basic approaches as described before. The Aggregation consists of basic map-reduce functions and computes the average numbers of passengers in the taxi rides in the specific windows.

## Latency

To measure the latency several ways exist. As Spark does not provide event time processing this benchmark basically calculates the processing time as latency. Therefore a timestamp is assigned for each tuples as soon as it enters the streaming system and another one before the resulting tuple leaves the system. As there are several input timestamps and just one output timestamp the timestamp of the latest incoming tuple is used for input. The difference between both is the latency.

Latency = timestamp\_result – maximum(timestamp\_1,…,timestamp\_N)

## Dataflow

The flow of the data and operations is following:

1. Read file and push tuples into Kafka topic
2. Streaming system reads from Kafka topic and assigns the first timestamp
3. Streaming system assigns tuples regarding their key into sliding windows
4. In each window with reduce functions the average number of passengers and the maximum timestamp is computed
5. The second timestamp is assigned
6. The result get printed

Figure 1: Dataflow model

**Data file**

**Kafka**

**Flink/Spark**

**Output**

**Timestamp**

**Timestamp**

**Latency**

## TODO Show example of aggregation

## System setup

The benchmark was run on a cluster with 10 nodes and 48 cores each.(TODO more specs) The setup was:

* 1 Master node
* 5 Worker nodes
* 1 Kafka node
* 1 Producer node
* 1 Zookeeper node

Flink was run with its default configurations while for Spark the backpressure mechanism had to be enabled for consistency.

## Experiment setup

The experiments were designed to analyze the influence of the window length, sliding length, and workload on the latency for both systems. Therefore 3 experiments were conducted, where each one parameter was changed at a time. The experiments were conducted at least twice to reduce some of the deviation.

Experiment “workload”:

* Window length: 3 seconds
* Sliding length: 2 seconds
* Workload: 50000 – 800000 tuples per second

Experiment “window length”

* Window length: 1 – 30 seconds
* Sliding length: 1 second
* Workload: 100000 elements per second

Experiment “sliding length”

* Window length: 30 second
* Sliding length: 1 – 30 seconds
* Workload: 100000 elements per second

For all experiments in Spark the batch size was set to 1 second. This is a bit rough as the performance of Spark is depending on this size, but to find the optimal size experiment would have done with a different batch size as there is no formula for the optimal measurement. Because this benchmark is more interested in the effect of the parameters than what could be the best latency, the value was set to 1 second for all experiments.

# Results

## Experiment workload

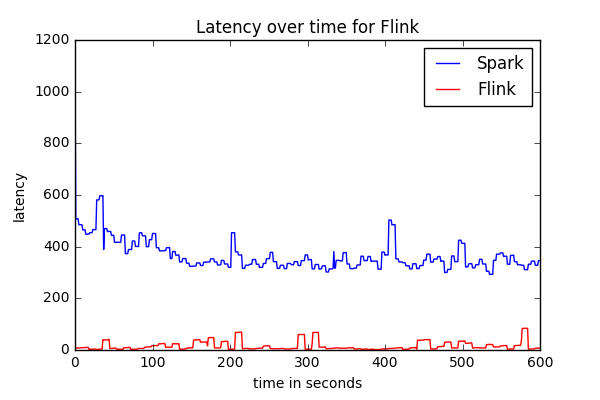
In this experiment the influence of the workload was analyzed. Basically, what to expect would be that with higher workload the number of elements to compute in each window would increase and therefore the latency would continuously increase as well.

Fig: 2

This figure is a scatter plot with one dot for each experiment and one regression line for each system. In contradiction to the previous expectations the latency stays constant for each system. Just a difference between both systems if observable. While Spark has a latency around 400 ms, Flink has a very low latency about 30 ms.

In figure 3 is the behavior of the latency in one experiment (the same as before) depicted. Here is shown that the latency stays for both system even for the same experiment approximately constant over time at around the same latency as in figure 2.

To explain the constant number of latency a closer look at the internally backpressuring of both systems is needed.



### Backpressuring

The backpressuring is a mechanism in a data streaming system where it is receiving data at a higher rate than it can process. If a situation like this happens it is more efficient to reduce the processing data than trying to process all data at once, because it probably results in a delay that the system cannot recover from easily. Both streaming systems have a built-in backpressure mechanism, although they behave differently. The problem with the backpressuring for the latency is that the latency in this benchmark is computed with as the difference of the data between entering the system and exiting it. And if the backpressure mechanism limits the number of elements that are going into the system for higher workload the latency gets independent from the workload and stays constant.

In figure 3 and 4 we can see how a streaming job run with and without backpressuring if the number of receiving tuples increase. In figure 3 are just 5 “blocks of tuples” incoming which also can all be processed. In figure 4 are 9 “block of tuples” coming from the source, but with due to backpressuring are only again 5 “blocks” entering the system. As the timestamps are assigned to the tuples just when they enter the systems, the latency stays the same even when the workload increases.

Processed tuples

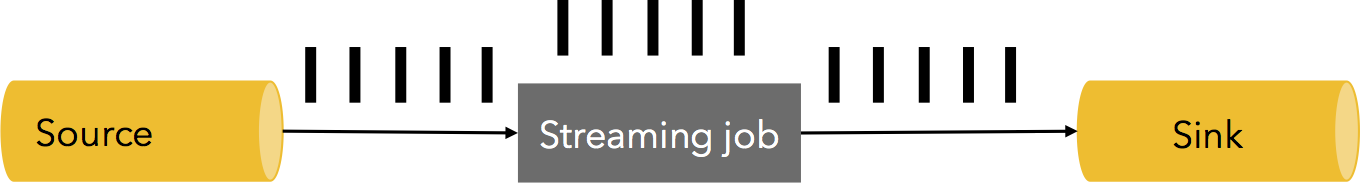


Figure3: Streaming job without backpressuring

Source: http://data-artisans.com/how-flink-handles-backpressure/

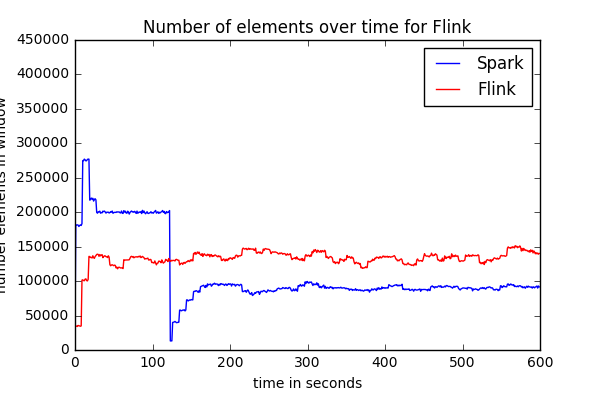
Processed tuples



Figure4: Streaming job with backpressuring

Source: <http://data-artisans.com/how-flink-handles-backpressure/>

In figure 6 are the number of elements that are processed in each window depicted to show how the backpressure mechanism operates. Here one experiment was run with window length of 25 seconds, sliding length of 1 second, and a workload of 300,000 elements per second. We see that Spark is in the beginning trying to keep up with amount of workload which leads into a delay of computation and it must reduce the input to nearly 0. Afterwards it increases the input slowly until it stays constant at around 90,000 elements. Flink in the opposite starts with less elements, but increases them faster. After a short period, it stays constant at about 130,000 elements.



With the learned knowledge of the backpressuring it makes sense to measure the average number of processed tuples for each window, which is basically the throughput, instead of the latency.

## Experiment window length

## Experiment sliding length

# Conclusion