Benchmark of windowed aggregations

By: Patrick Lehmann, TU Berlin

Mentor: Jeyhun Karimov

Date: 22.01.2017

Table of contents:

1. Motivation/Goals

Due to the various number of different streaming engines like Apache Flink [1], Apache Spark [2], Apache Storm [3], Apache Samza [4], and Apache Apex [5], 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. To figure out where each system has its advantages and disadvantages it is from importance to benchmark those. But not only to compare each system a benchmark is important, but also to analyze how one system´s performance behave regarding different parameters. Because many factors influence the performance this benchmark focuses on windowed aggregations. Therefore, it will analyze

[1] https://flink.apache.org/

[2] http://spark.apache.org/

[3] http://storm.apache.org/

[4] http://samza.apache.org/

[5] https://apex.apache.org/

parameters like the window length, the sliding length, and the workload. To measure the performance the latency will be computed.

2. Project setup

To generate the data stream, a Taxi Ride file [6] 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 [7], 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. The aggregation also uses sliding window which are combined with a “keyBy” function and different keys to achieve parallelization.

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 tuple 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 in the window is used for input. The difference between both is the latency.

Latency = $\text{timestamp}_{\text{\_result}}$ – maximum ($\text{timestamp}_{\text{\_1}}$, …, $\text{timestamp}_{\text{\_N}}$)

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

[6] http://dataartisans.github.io/flink-training/exercises/taxiData.html
[7] https://kafka.apache.org/

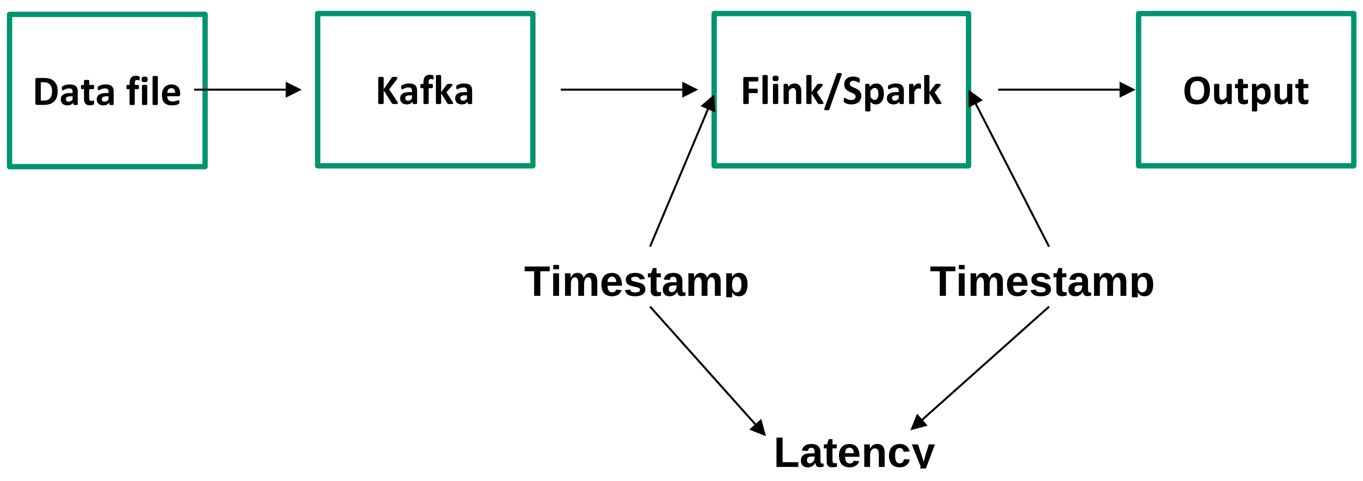

Figure 1: Dataflow model

System setup

The benchmark was run on a cluster with 9 nodes and 48 cores each.

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 parameter was changed at a time. Most experiments were conducted several times 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: 300000 elements per second

Experiment “sliding length”

- Window length: 30 second

- Sliding length: 1 – 30 seconds
- Workload: 300000 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. To find the optimal size, experiments should have done with different batch sizes as there is no formula for the optimal number, which causes a lot of recomputations. 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.

3. 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.

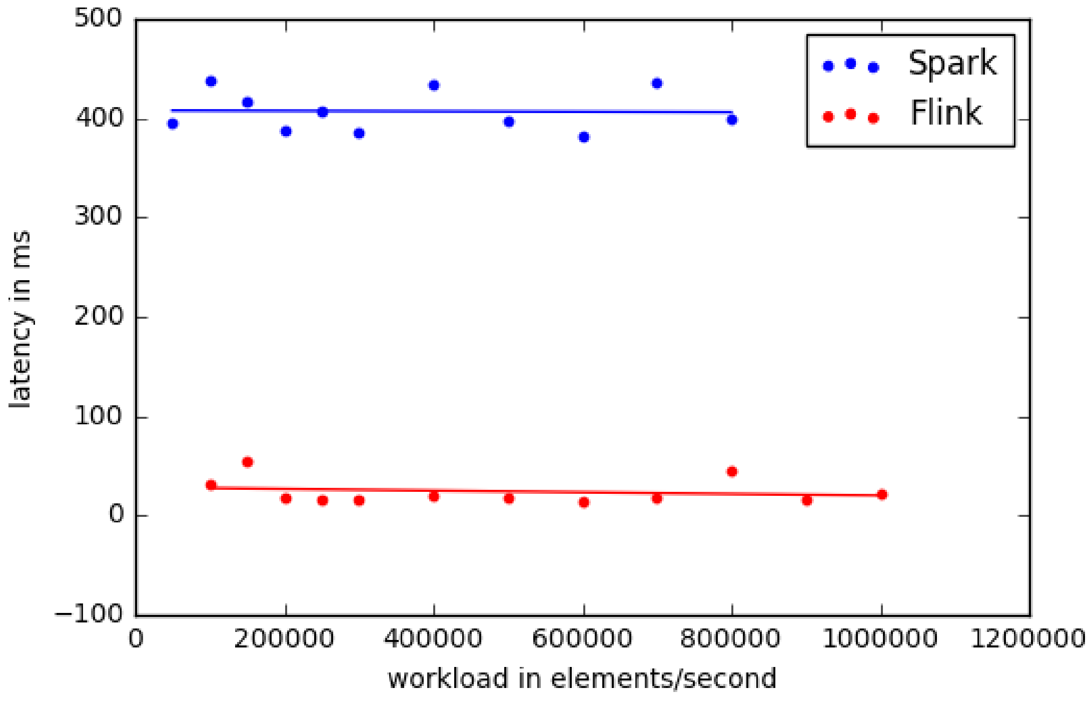

Figure 2: Workload for window length = 3s and sliding length = 2s

Figure 2 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 configuration 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.

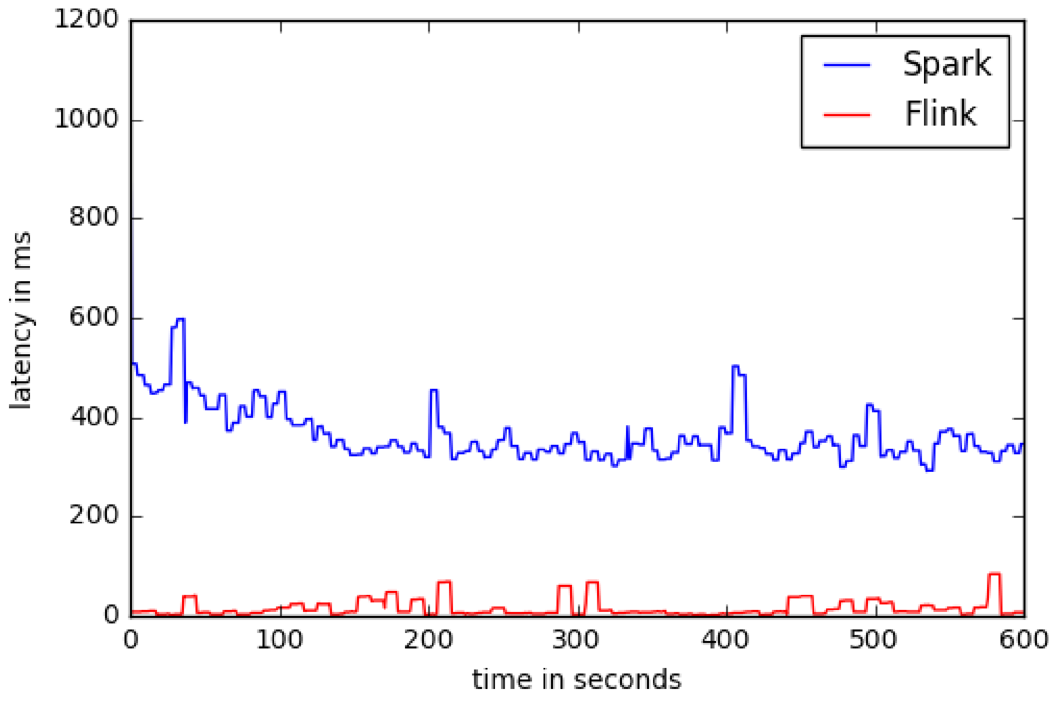

Figure 3: Latency over time for Flink and Spark

Backpressuring
The backpressuring is a mechanism in a data streaming system which operates when its receiving rate of data is 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 computation of the latency is that in this benchmark it is computed as the difference of time 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 runs 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 because of the backpressuring only 5 “blocks” can enter the processing of 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.

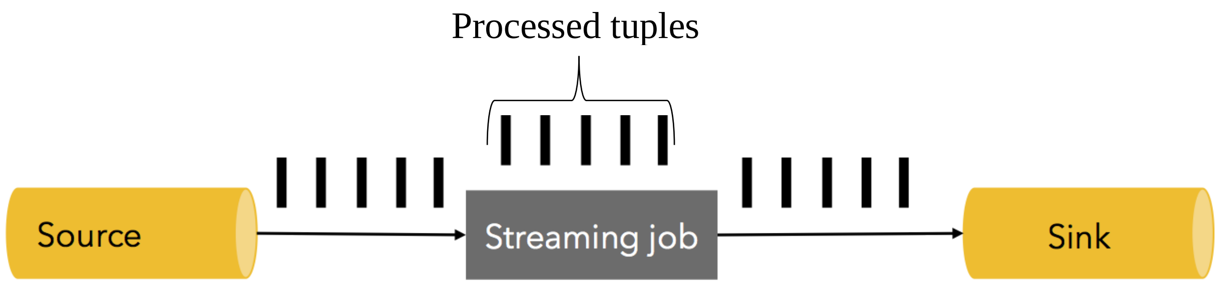

Figure4: Streaming job without backpressuring [8]

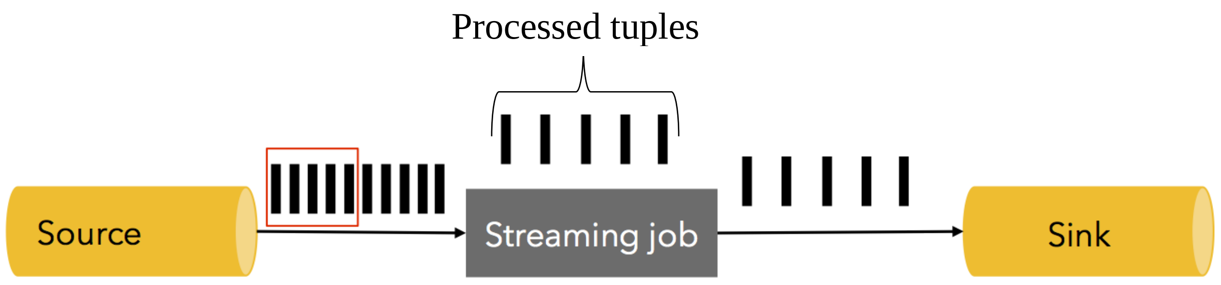

Figure5: Streaming job with backpressuring. Source: [8]

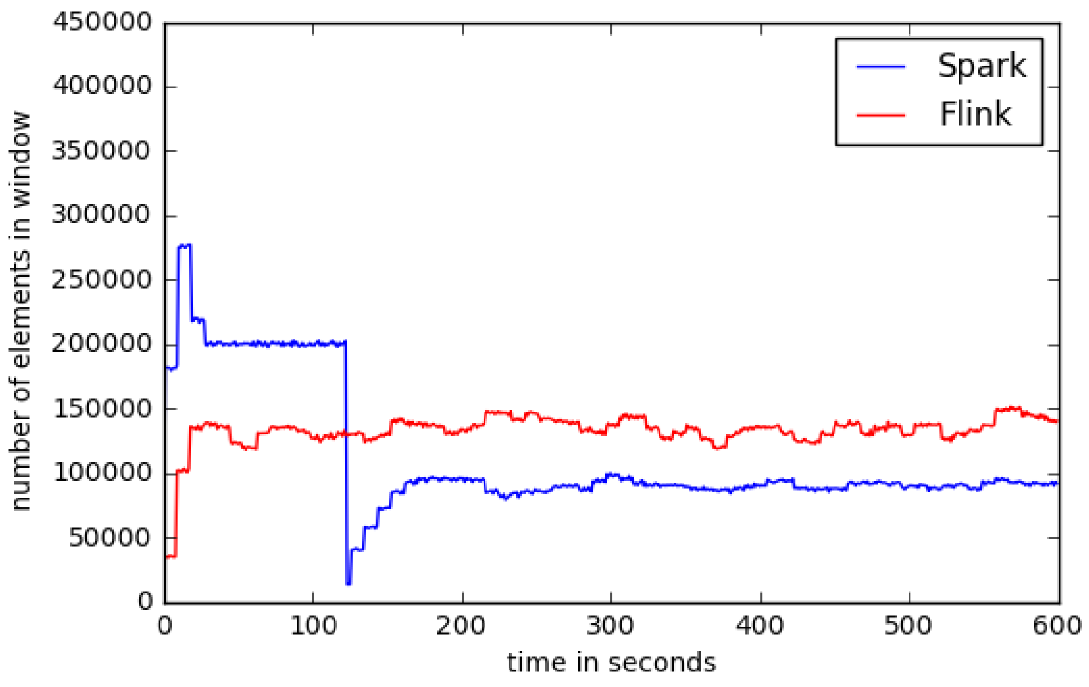

Figure 6: number of elements over time for Spark and Flink

[8] *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. Therefore, one can see that for this configuration Flink performs better with an average about 130,000 over Spark with just 90,000 elements.

To overcome the resulting problems of measuring the latency several methods are possible. The simplest would be to disable the backpressuring. For systems like Spark it is disabled by default, but for other systems it might not be possible to disabled the mechanism. Also in real world applications, the backpressuring it normally enabled so it would not make much sense to make a benchmark with disabled backpressuring. Another way would be to measure the latency differently. One could think of using timestamps at the creation time of the elements. Therefore, the time the elements wait to processed would be included as well. A disadvantage for this method is that Spark for example does not provide event time processing and cannot guarantee that the elements will be processed in the same order like they were created. A third way could be to find another metric to measure the performance which can not only show off differences between the systems, but also the behavior over time for one system. Such an indicator was used in this benchmark and will be explained in the following part.

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. Figure 7 shows an example for that. Again, these graphs seem to be constant. This happens, because with a high workload the systems´ backpressuring limits the input to the computed optimal number and the number of elements that are getting processed are getting independent of the workload. On one hand this shows good which systems has a better throughput, but on the other hand it does not exactly tell how these systems perform regarding to the specific parameter. Hence, a different metric is needed to show also the influences of the parameters. As the possible input increases with higher workload, one might be interested in the percentage of the number of tuples that could be processed. This is shown in figure 8. This figure shows that Flink can process with lower workload about 80 % of the incoming tuples, but is continuously decreasing as the workload increases. Spark performs much worse as with low workload it could just process about 40 % of the workload and with a workload of 200,000 elements the performance rate is lower than 20%. Although the processing rate of Flink is decreasing faster, relatively they are decreasing approximately the same.

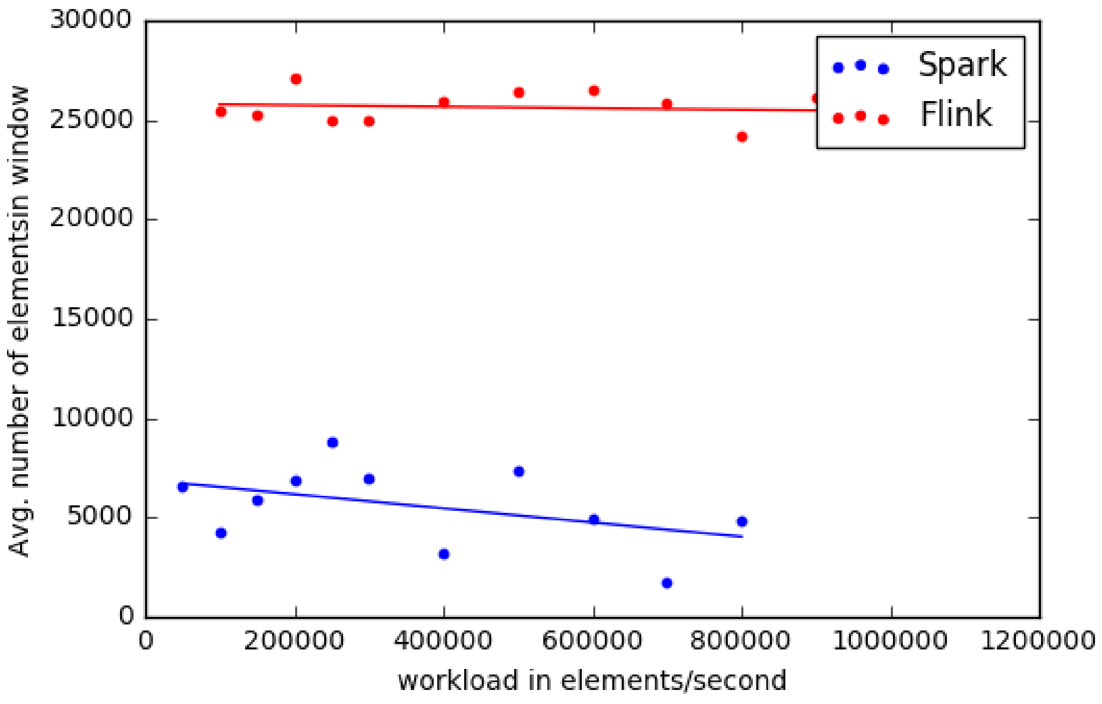

Figure 7: Average number of elements for experiment "workload"

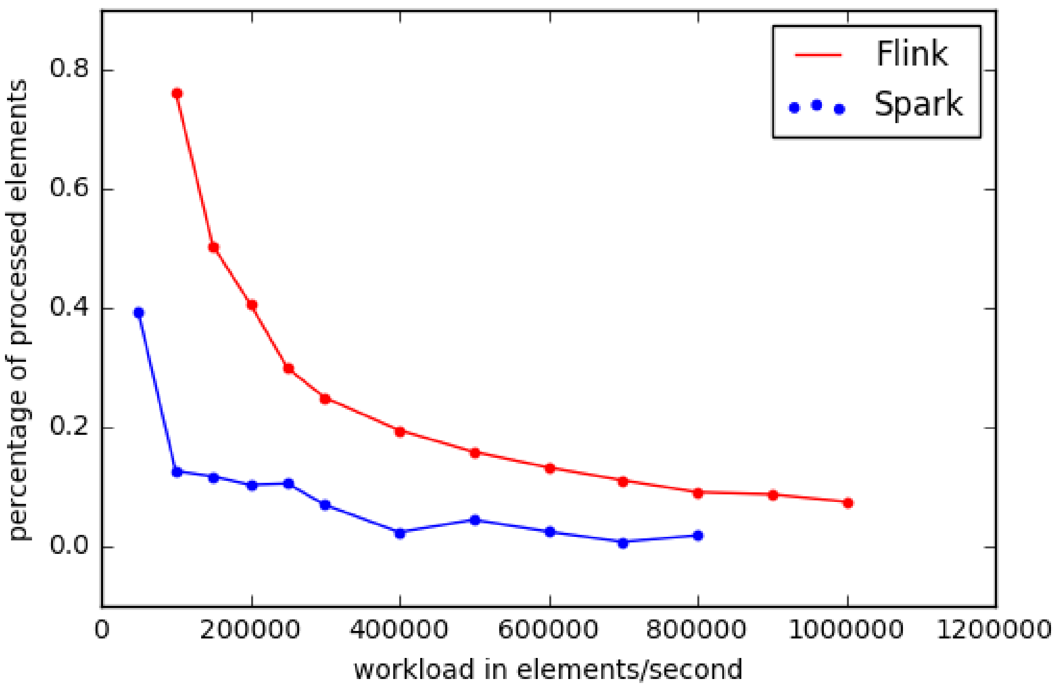

Figure 8: Percentage of processed elements for experiment "workload"

Therefore, in general it was shown that the latency is not sufficient to do a benchmark for streaming systems if the measurement is based on processing time and they have backpressuring enabled as this mechanism fixes the latency to be approximately constant. Therefore, it is helpful to analyze the throughput as well. In this benchmark for both values Flink was performing better, but both measurements stayed constant over time. The percentage of elements that were processed gave a better view on the behavior regarding the workload. While Flink was performing way better for smaller workload, it was also decreasing faster, but performed better than Spark for the whole experiments. Spark could not keep up with lower workload and performing terrible for workload over 100,000 elements per second.

Experiment “window length”

In this experiment the influence of the window length was analyzed. Therefore, an experiment was conducted that had a workload of 300,000 elements per seconds, a sliding size of 1 second, and the window length was changed from 1 second to 30 seconds. One might expect there would not be a large influence of the window length, because the number of new elements that would come into the streaming system is determined by the sliding length, which stayed constant for this experiment. On the other hand, the number of elements that are in the windows should increase with larger window length and consume therefore more memory which could affect the performance.

Figure 9 shows the average number of elements in the windows for increasing window length. Although for both systems the number of elements are increasing it is obvious that their slopes are different in height. Flink seems to better handling the increasing window length as Spark does. But even for Flink one can already see the window length influences the performance as with doubling the window length the number of elements are not doubling. This is shown better in figure 10 where the percentage of processed elements is depicted. Both systems have a negative slope which means they are influenced by the window length. The slope of Flink is however not as much affected as the one of Spark is. To note here is the outlier of Spark for the window length of 1 second. As the sliding length was 1 second as well the window was in this experiment a tumbling window. For those tumbling windows Spark seemed to perform significantly better than for sliding windows in other experiments as well. This might explain that outlier, but further researches would be required to confirm this.

In summary, it was shown that the window length has a slight influence on Flink´s and a larger one on Spark´s performance.

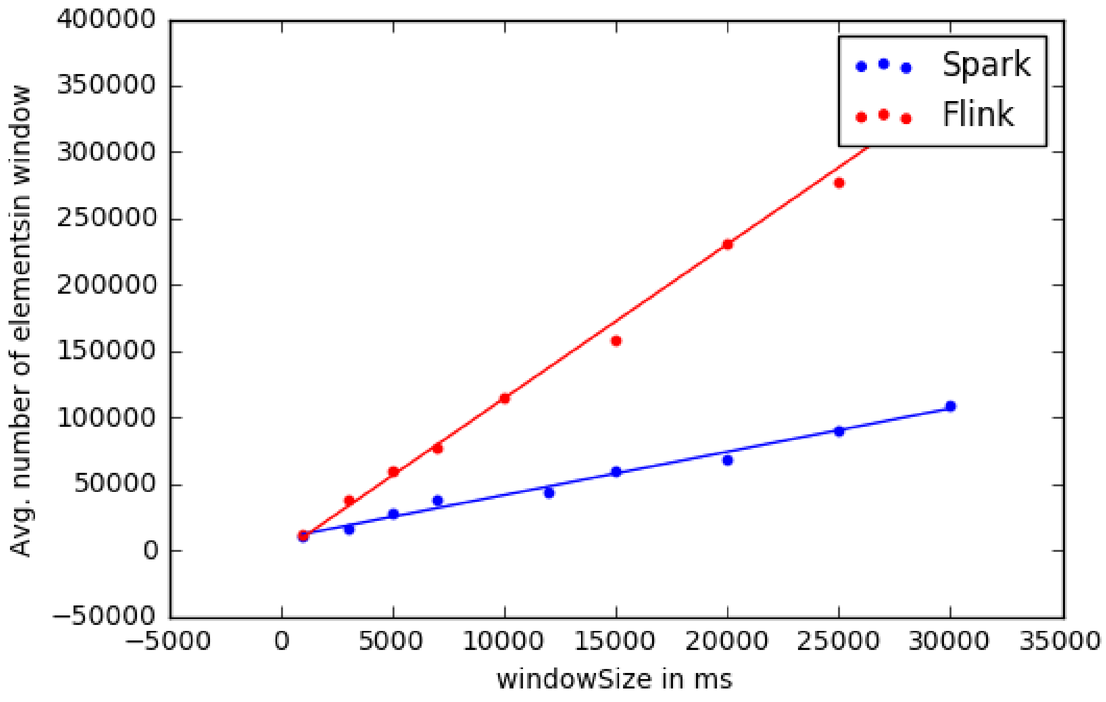

Figure 9: Average number of elements for experiment window length

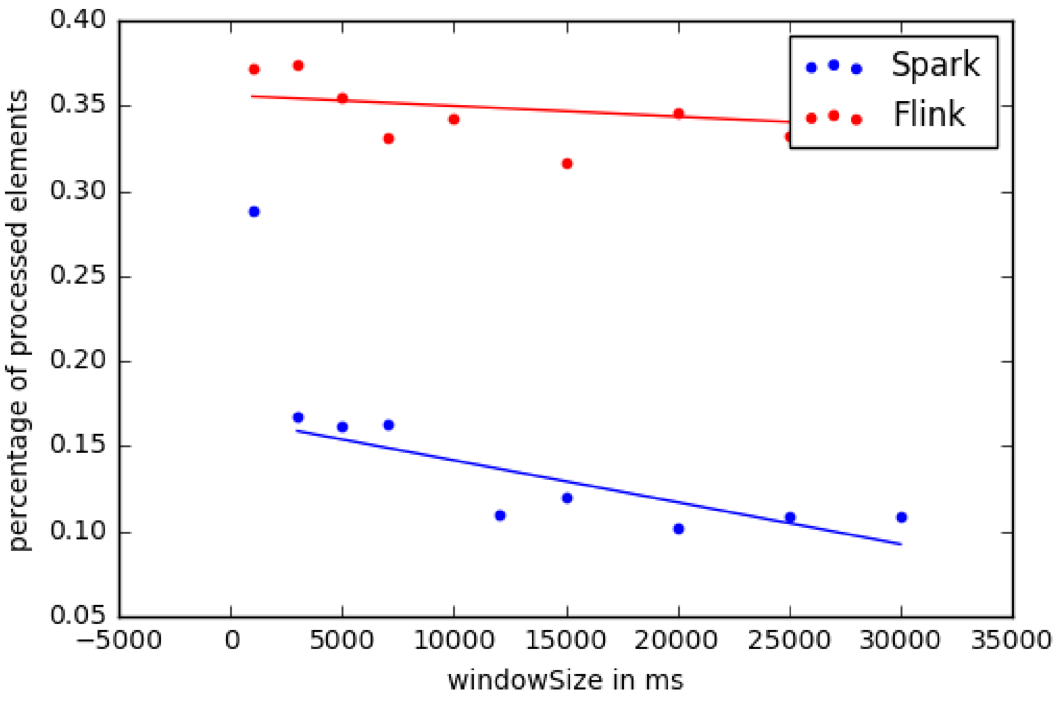

Figure 10: Percentage of processed elements for experiment window length

Experiment "sliding length"

For the sliding length, an experiment was conducted with a workload of 300,000 elements per second, a window length of 30 seconds, and a sliding length from 1 second to 30 seconds. In figure 11 is the percentage of processed elements shown. This graphs would be the same, except the scale, for the measurement of average elements as the input rate of all experiments was the same. The latencies were again constant, because of the backpressuring.

The behavior of Flink´s processing rate is approximately constant with a slight negative tendency. That tendency might be explained by some variations. Spark´s graph just performing slightly better for small sliding length. Afterwards the processing rate is with under 5 percent performing poorly. For larger sliding sizes the performance could can better with an increasing batch size as this was or all experiment constant with 1 second. Compared to Flink performance Spark´s performance would even with dynamic batch sizes much worse. The large difference can be explained by the high workload and window length. As seen in previous experiment both parameters decrease Spark´s performance larger. For this experiment, they seem to add their effects which would explain the large difference between both systems. To note again, is an outlier in Spark´s performance at 30 seconds sliding length. As this is the same size as the window length, the underlying window is a tumbling window. Like in the previous experiment it seems that Spark´s performance is better for tumbling windows.

Generally, it seems as the sliding length does not have a big influence on the performance for both systems.

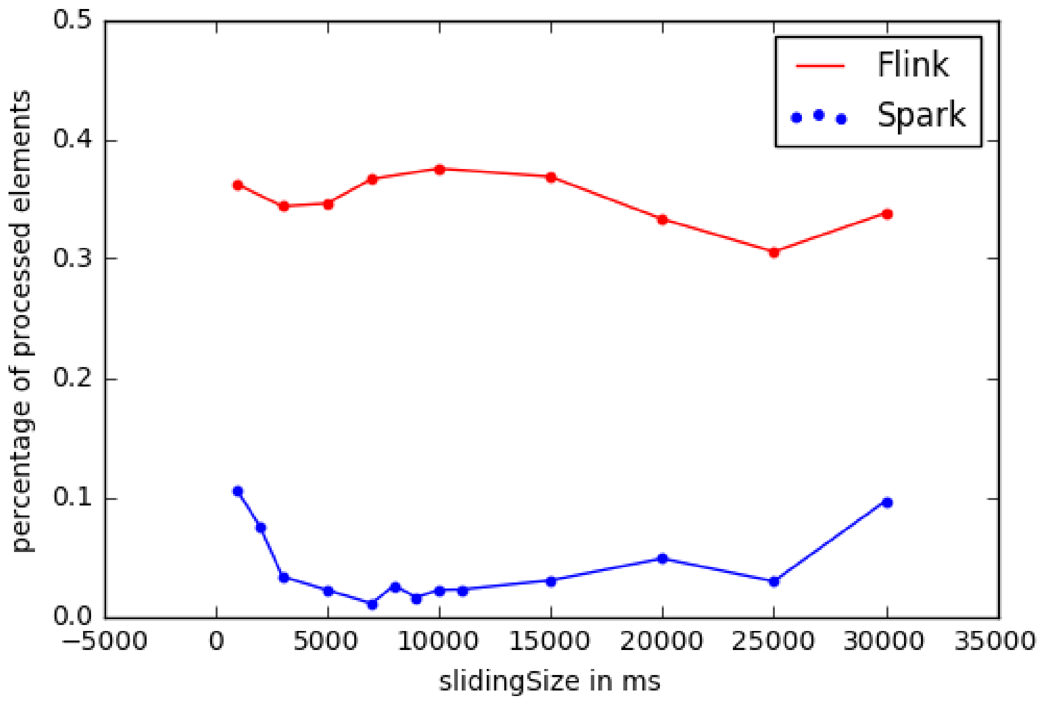

Figure 11: Percentage of processed elements for experiment sliding length

4. Conclusion

In summary, it was shown that the latency is not the only measurement needed to explain the behavior of the systems. This happens if the backpressure mechanism is enabled, because this mechanism limits the receiving rate and the measurement of the latency in this benchmark was using the processing time. Hence, the number of elements to process and the latency stay approximately constant. Several methods to overcome this problem were briefly discussed. Therefore, another metric was used to indicate the performance. This was the rate of elements that were processed in each window. For example, this could show that one system could process 50 percent of the receiving elements for a specific configuration. With that metric the behavior of the throughput regarding the parameters could be shown.

For the experiment “workload” both systems´ percentage was decreasing with increasing workload as seen in figure 8. Flink´s processing rate was generally better than the one of Spark, although it seems that the influence of the workload on the performance of Flink is larger, because it is decreasing faster. As in comparison to Spark the graph was starting higher, the influence on both graphs therefore is relatively nearly the same.

The experiment “window length” was showing in figure 10 a kind of same behavior for both systems. Not only Flink´s processing rate was again much higher than Spark´s, but also the influence of the window length does not have such a big influence on the performance as it has for Spark. Spark´s processing rate was much more effected as the one of Flink.

The experiment “sliding length” as depicted in figure 11 shows not such a big influence as the other parameters. For Flink there might be a very little tendency, that also could be explained by some variation within the experiments. Spark has had in the experiments just a decreasing performance for lower sliding lengths. Afterwards there was not a change in the performance except when the sliding length equaled the window length resulting in a tumbling window which seemed through all experiments, as discussed in the previous section, to have a better performance. Despite that it seems that there is no significantly influence of the sliding length on the performance of the systems.

To compare the influences of the parameters is hard as the workload has a different metric as the other parameters. Generally, it was shown that the workload and the window length have a bigger influence than the sliding length.