A Comparison of Distributed Stream Processing Systems for Time Series Analysis

Melissa Gehring[1], Marcela Charfuelan[2], Volker Markl[3]

Abstract:

Given the vast number of data processing systems available today, in this paper, we aim to identify, select, and evaluate systems to determine the one that is better suited to use in conducting time series analysis. Published studies of performance are used to compare several open-source systems, and two systems are further selected for qualitative comparison and evaluation regarding the development of a time series analytics task. The main interest of this work lies in the investigation of the *Ease of development*. As a test scenario, a discrete Kalman filter is implemented to predict the closing price of stock market data in real-time. Basic functionality coverage is considered, and advanced functionality is evaluated using several qualitative comparison criteria.

Keywords: Stream data; Stream processing; Time series analysis; Predictive analytics

1 Introduction

Today's batch and stream processing systems possess vastly different characteristics and are designed to tackle diverse classes of problems. Batch processing systems (BPS), such as MapReduce-based systems, are well-suited for querying stored historical data (a.k.a. data-at-rest). In BPS, data processing is efficient, and administration overhead is minimal (relative to real-time processing systems [SG17]), but they cannot face the constraint of real-time. In contrast, stream processing systems (SPS) are able to process data in real-time (a.k.a. data-in-motion), which is necessary since streaming data is unbounded. Stream processing is performed at the event, window, or micro-batch level [SG17].

Time series data (TSD) are comprised of a series of measurements (e.g., stock market data, medical data, meteorological data [BKF17]), that are taken at a given time-scale (e.g., second, minute). Traditionally, time series management systems (TSMS) and time series data bases (TSDB) have been used to process and store TSD, as discussed in recent surveys (e.g., TSMS [JPT17], TSDB [BKF17]). The frameworks/systems analyzed in these surveys offer querying and data storage capabilities and additionally support some data analytics

[1] Technische Universität Berlin, FG DIMA / Fakultät IV, m.gehring@campus.tu-berlin.de
[2] DFKI / Technische Universität Berlin, marcela.charfuelan@dfki.de
[3] Technische Universität Berlin / DFKI, volker.markl@tu-berlin.de

and may include stream processing capabilities. However, in general, stream processing is not a requirement for TSMS and TSDB. Instead, these systems are designed to handle historical TSD and are ill-suited for real-time processing. Additionally, the majority of these systems are unable to perform advanced analytic tasks, such as prediction, forecasting and similarity search. Recently, real-time SPS, capable of performing stream and batch analytics, have emerged. Thus, we seek to conduct a practical evaluation of these state-of-the-art systems for time series analysis. For our experiments, we implemented a discrete Kalman filter to predict the closing stock market prices in real-time. In this paper, the focus is on the examination of the *Ease of development* of the various implementation steps. Qualitative criteria are defined in order to compare the implementation steps of the predictive task in a streaming test scenario.

The paper is organized as follows. In Section 2 we discuss the selection and evaluation of two open-source SPS for time series analysis using published studies of performance. The qualitative comparison criteria are also defined within this Section. In Section 3 we describe the test scenario and summarize the qualitative assessment that we underwent. In Section 4 we present our conclusions.

2 Comparison Methodology

2.1 Stream Processing Systems

Several criteria are available to characterize SPS. An active field of research is the performance analysis, applied to different systems using quantitative measurements, such as latency and throughput. In this paper, the latest versions of the one-at-a-time based SPS **Apache Storm**[4], **Apache Flink**[5], **Apache Samza**[6] and the microbatch based systems **Storm Trident**[7] and Spark Streaming of **Apache Spark**[8], are compared utilizing several published studies of performance [Wi16, KKR15, Ch16, Ka18]. In Table 1 the characteristics of the previously-mentioned systems are summarized. According to the Yahoo Streaming Benchmark [Ch16], Flink and Storm offer lower latencies, whereas Spark is able to handle higher throughputs, while having somewhat higher latencies. This finding confirms the common statement that there is a difference between micro-batch and one-at-a-time based SPS. According to the Karimov et al. Benchmark [Ka18], (1) Spark is better suited to overcome streams containing skewed data, (2) Flink and Spark are very robust to fluctuations in the data arrival rate in aggregation workloads, (3) Flink is best at handling fluctuations in the data arrival rate on join queries, (4) Flink provides the lowest average latency, (5) Spark (with higher average latency) manages to bound latency best, and (6) Flink has higher throughput with a low latency, in use-cases containing large windows.

[4] http://storm.com
[5] http://flink.com
[6] http://samza.com
[7] http://storm.com/trident
[8] http://spark.com

	Storm	**Storm Trident**	**Spark Streaming**	**Flink**	**Samza**
Processing model	one-at-a-time	micro-batch	micro-batch	one-at-a-time	one-at-a-time
Delivery guarantee	at-least-once	exactly-once	exactly-once	exactly-once	at-least-once
Backpressure mechanism	yes	yes	yes	yes	buffering mechanism
Ordering guarantees	no	between batches	between batches	no	within stream partitions

Tab. 1: Comparison of stream processing systems adapted from [Wi16]

Choosing a processing model always means trading off between latency and throughput. Latency must also be traded against other desirable properties, such as message delivery guarantees and ease of development, as they increase the per data-item overhead (messaging and state replication). Rich processing guarantees at the same time make a system more reliable. Thus, a variety of characteristics can influence the selection of a suitable framework. Although it is still challenging to select an appropriate stream processing system considering performance issues and reliability, two frameworks provide great support. Apache Spark and Apache Flink stand out to bring better performance, unified with the most advanced features in comparison to Storm, Storm Trident and Samza.

Support for	**Storm**	**Storm Trident**	**Spark Streaming**	**Flink**	**Samza**
Languages	J, P, L, R	J, S, C	J, S, P	J, S	J
Event-time	no	no	yes	yes	no
Watermark, allowed lateness, trigger	no	no	partly	yes	no
Windows, joins, filter, aggregations, etc.	no	yes	yes	yes	yes
Stream SQL	no	yes	yes	yes	no

Tab. 2: Functionality across stream processing systems (J: Java, S: Scala, P: Python, L: Perl, R: Ruby, C: Clojure)

Unfortunately, the evaluation criteria did not further analyze the *Ease of development*, which for a beginner in working with SPS is an important property. The usability of SPS can be increased, for example, by offering richer language support as well as the availability of higher-level APIs with support for common functions, like windowing, joins, filtering, aggregations, or stream SQL support. Furthermore, usability is enhanced if advanced features, like event-time processing, watermarking, and triggers are offered. Table 2 summarizes key features commonly associated with the different SPS. Storm supports a wide range of languages but does not provide a high level-API. Samza and Storm Trident offer some advanced features like support for windowing, filtering, etc. but lack of event-time support. Flink and Spark support these advanced features and also provide an API to use SQL within streams. Storm Trident does as well, but a lack of event-time support and active support by the community [Ka18] exist. Spark took a big step forward by integrating event-time support with Spark Structured Streaming Version 2.1.

Both Apache Spark and Apache Flink stand out, achieving higher performance and offering the most advanced features. Thus, these two systems are selected for qualitative comparison.

2.2 Qualitative Criteria

Our selection of criteria is inspired by the work of Armstrong [Ar01], where various quantitative and qualitative criteria are considered when selecting among various time series forecasting methods. *i) Ease in using available data*, *ii) Ease of use*, *iii) Ease of implementation* and *iv) Flexibility* are among the top ten criteria. These qualitative aspects are affected by the selection of implementation framework. Therefore, in our test scenario, we conduct our qualitative analysis on the basis of these criteria. The following measures are defined for rating the first three criteria: **Extent**, **Simplicity** and **Documentation**. The fourth criterion, *Flexibility*, is considered to evaluate the implementation of more advanced functions in the prediction task. Yet, another measure, the **Adaptability**, is defined to rate the systems regarding the criterion of *Flexibility*. Table 3 shows the various rating levels defined for the measurements and their corresponding meaning.

	Basic functions			*Advanced functions*
Rat.	**Extent**	**Simplicity**	**Documentation**	**Adaptability**
++	Additional features available	Simple and understandable concept. Automated functionality.	Good access and high quality.	Adaptation integrable with low user side implementation overhead
+	Function and essential features available	Clear concept. Reasonable user side implementation overhead.	Access insufficient.	Adaptation integrable with reasonable user side implementation overhead.
0	Function available.	Concept is not clear or disproportionate user side implementation overhead.	Mentioned in documentation	Adaptation theoretically possible but disproportionate implementation overhead.
-	Function not available	Function not available	Documentation not available	Adaptation not possible/ integrable.

Tab. 3: Criteria and Rating for qualitative comparison of *Ease of Development* in Spark and Flink

3 Test Scenario and Qualitative Comparison

Test Scenario Pipeline. In SPS, sources and sinks are typically employed. Data is consumed from a source, then processed within the system before being sent to a sink. The pipeline shown in Figure 1 is employed in the test scenario. Flink and Spark consume their data from Apache Kafka [9], which is a distributed streaming platform serving as a message broker. A Kafka topic is created, and data is then sent to this topic, where it is persistently stored. The SPS can subscribe to the topic and will then always directly get the newest data published by Kafka in the particular topic. The prediction task in the test scenario requires the implementation of various tasks or functions in several steps. Table 4 shows the main steps organized in two groups (basic and advanced), the corresponding comparison features and the qualitative measurements and ranking for each step. For visualization and verification purposes, the pipeline will be extended by a persisting instance, i.e., the special TSDB InfluxDB [10]. InfluxDB can be used as source for Grafana [11], a modern time series visualization tool.

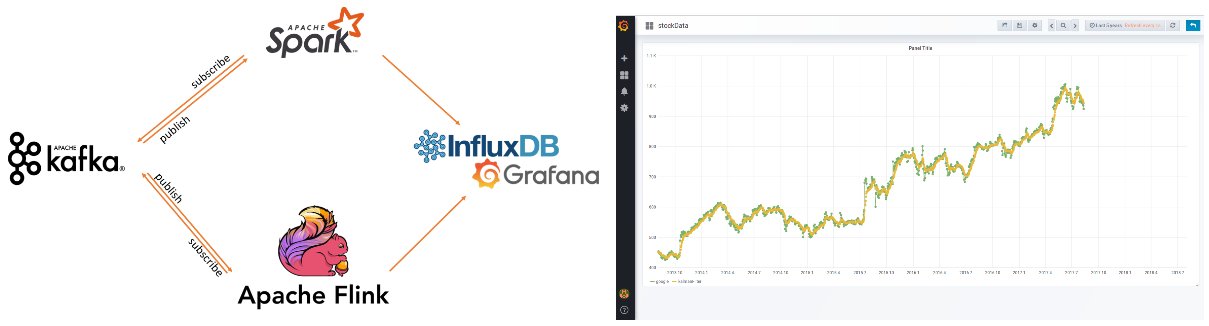

Fig. 1: Stack for test scenario implementation and example of stock data visualization

Test Data. Stock market data time series will be used for the prediction task. The used data set available at Kaggle[12], contains data from the S&P 500 (i.e., Standard & Poor's 500) index. The dataset contains historical data over a five-year period (2012-08-13 through 2017-08-11) across all current 500 companies listed on the S&P 500 index.

3.1 Qualitative Comparison

Step 1 - Setup. Setting up the pipeline is not challenging for either system. Although both systems support several APIs for various kinds of data processing, batch and streaming, Flink is more focused on streaming, which is also reflected in the documentation. In Spark the user needs to take a deeper look to find the right documentation and information for setting up the streaming environment.

[9] http://kafka.com
[10] http://influxdb.com
[11] http://grafana.com
[12] https://www.kaggle.com/

	Simplicity	**Documentation**
Flink	+ Different libraries need to be included either working in Java or Scala	++ Short and clear documentation. Easy to find as directly provided in Download area.
Spark	+ Different libraries need to be included, as wide range of libraries exist, all offering interesting functions; there is no centralized library.	+ Short and clear documentation. Harder access due to spread of dependencies, they are located in the documentation areas of different APIs.

Step 2 - Time handling. Handling time is a very important aspect to consider in developing streaming applications. Very often it is desired to process data using the event-time, the time when the event occurred, instead of the processing-time, the time of the machine when the data is processed. Although not all SPS provide support for this type of time processing, both Flink and Spark do. Flink's event-time support has been available for quite some time. It is mature and further offers a wide range of additional features. Event-time support in Spark is a drawback, since it is still a new feature. The integration of event-time support is not in the Spark Streaming library, but rather is in the Spark Structured Streaming library.

	Extent	**Simplicity**	**Documentation**
Flink	++ Wide range of features available (watermark, allowed lateness, triggers)	+ Complex concept for event-time support. The user needs to define parts of the concept (TimeStampExtractor) himself or can use predefined ones (not automated).	++ Very clear and helpful documentation with examples and detailed description and easy to find.
Spark	+ Basic event-time support available and watermark configurable.	+ Easy and understandable concept, the user can easily group by window due to saving event-time within columns, but the system breaks (transformation to other data format is necessary)	0 Hard access due to lack of remark about event-time support within Spark Streaming documentation, just in Spark's Structured Streaming documentation (other library). Documented functions are then not directly applicable since other data structure has to be used.

Step 3 - Stream Source Connector. As Kafka is a widely adopted source for SPS it is used to provide access to event streams. The integration is easy to realize within both systems.

	Extent	**Simplicity**	**Documentation**
Flink	++ Wide range of features available (deserializer, offset control, etc.)	++ Easy and clear concept for Kafka integration, especially the stream creation (2 actions for configuration, 1 for stream creation as a DataStream).	++ Very clear and helpful documentation with examples and detailed description and easy to find.
Spark	++ Wide range of features available (deserializer, offset, etc.)	++ Easy and clear concept for Kafka integration, especially the configuration (1 action for configuration, 1 complex action for stream creation as a DStream)	+ Clear and helpful documentation but missing examples and description for advanced features. Confusing location of documentation part.

Step 4 - Preprocessing. The functions necessary to apply preprocessing are very basic ones within a stream processing system, as the preprocessing is a step that always needs to be applied. There is almost no difference between Flink and Spark in this step. The functions make it possible to implement user-defined functions that can process custom data items.

	Extent	Simplicity	Documentation
Flink	++ Very flexible. Expression and self defined functions can be used within transformation functions.	++ Easy concept, easy to use, understandable application. Just 1 action necessary for integration of self defined functions.	++ Good access. Detailed documentation of concepts with understandable examples and explanations.
Spark	++ Very flexible. Expression and functions can be used within transformation functions.	++ Easy concept, easy to use, understandable application. Just 1 action necessary for integration of self defined functions.	+ Good access. Detailed documentation of concepts and understandable examples available, but missing explanations of the examples.

Step 5 - Stream Processing. Stream processing operations are typically applied using event-time. A basic operation is the implementation of sliding windows and transformations afterwards. Within this step, the implementation of a moving window average is compared. As the window is applied using event-time, Spark's new Structured Streaming API needs to be used, which makes operations on multidimensional data possible. Stream functions are easy to implement, as SQL-based operations on streams are available. In Flink the concept is clear, but implementations need to be done manually due to missing support of multidimensional aggregations in predefined functions.

	Extent	Simplicity	Documentation
Flink	+ No predefined function available for advanced features, in form of multidimensional aggregation on windows, but realizable through UDFs. Sliding window and event-time support available.	+ High complexity of the concept to implement UDFs. 2 actions necessary (1 implementation of user defined function and 1 for application of user defined function).	++ Good access. Detailed documentation of concepts with understandable examples and explanations.
Spark	++ Advanced features available. Aggregation of multidimensional points on windows realizable through SQL based aggregations. Sliding window and event-time support available.	++ Very easy and understandable concept due to SQL based operations. 1 user action necessary for operation.	++ Good access. Detailed documentation of concepts with understandable examples and explanations.

Step 6 - Time Series Analysis. In a first step of the TSD analysis, a version of the discrete Kalman filter is implemented in both systems. For simplicity, assumptions made by Welch et al. [WB01] were used. The code is adapted to fit the use case in the form of processing stock data in streaming fashion. In the second step of the TSD analysis, an online evaluation is implemented. To do so, an error calculation has to be integrated in the Kalman filter algorithm that compares the previous predicted value with the current actual value.

	Adapting to stock market use case	**Integrating error calculation**
Flink	++ Due to easy implementation as a function applicable as a flatmap transformation on stream the adaptation to the use case was easily realizable, just the input and output had to be customized as well as the calculation to be using closing price.	++ Integration without any issues due to easy implementation of the algorithm as a flatmap function.
Spark	+ The implementation of the Kalman Filter is using a more complex concept. The data source needed to be adjusted so that a DStream could be processed. Due to the stream coming in as batches that adaptation resulted challenging. Furthermore processing of stock data items had to be enabled, as well as outputting predicted items. Calculation had to be adjusted to work on closing price. These adaptations were possible to realize without problem.	++ Easy integration of error calculation into returned object.

Step 7 - Evaluation. Two means of evaluation are considered, i.e., an online evaluation during stream processing and another, conducted after collecting and extracting processed streaming results. Both Flink and Spark, provide different possibilities to analyze data resulting from the application. Both provide the functionality of applying SQL to the data after transforming it to a new data format. The results of the queries were not possible to extract, so further analysis was not possible. Another option tested was to apply aggregation functions on the streams directly in Flink and on the RDDs in Spark. In Flink, the results were calculated continuously on the entire stream; in Spark the calculation only considered items within the RDD. Due to these problems, sinking the results to a .csv file was preferred. This is possible in Flink, but not in Spark due to the partitions made for each RDD.

	Perform evaluation
Flink	+ Wide range of possibilities to perform evaluation available: great support for evaluation directly on stream of errors and export to csv easily possible. Challenging when converting stream to a data set and trying to export the results achieved applying SQL.
Spark	0 Wide range of possibilities to perform evaluation available, but none of them could be applied correctly due to problems handling the RDDs.

Step 8 - Visualization. To integrate the visualization using InfluxDB and Grafana a connector is necessary. InfluxDB connectors exist for both Flink and Spark. The integration of the connectors was analyzed according to their *Adaptability*, considering the integration in general, the customization necessary to write stock data into the data base and the possibility of adding other sinks to other streams (at various stages of TSD processing). The connector available for Flink achieved very good results for all these criteria, Figure 1 shows a screenshot of this visualization. Unfortunately, the connector available for Spark could not be integrated due to missing information and documentation.

	Integration of visualization connector
Flink	++ Available connector provides high flexibility, easy adaptation to the use case, so that custom data formats are accepted. Very easy handling and integration to the use case (Just adding customized sink to DataStream).
Spark	- No easy-to-integrate connector available.

4 Conclusion

Table 4 summarizes the results of Section 3. Generally, both Flink and Spark, provide qualitatively high basic functionality, necessary for the development of basic streaming applications. When developing advanced applications for TSD, Flink appears to be better suited than Spark, since the programming abstraction complexity is lower in Flink. Furthermore, Spark requires the use of a wide range of frameworks to incorporate diverse functionality and requires transformations across several APIs.

The DataStream API provided by Flink is straightforward and thin. Flink's programming abstractions for streaming are simple and ease the development within advanced tasks. Spark's Streaming API cannot directly be compared to Flink's DataStream API, as the essential event-time support is not integrated in the Spark Streaming API. A fair comparison would be using Spark's Structured Streaming API. Due to the relatively new API, many advanced algorithms and implementations are not yet available for Spark Structured Streaming from contributors. This was also an issue during the implementation of the test scenario. Implementations of the Kalman filter were only available for Spark Streaming.

Step	**Criteria**	**Flink**	**Spark**
1 - Setup	Simplicity	+	+
	Documentation	++	+
2 - Time Handling	Extent	++	+
	Simplicity	+	+
	Documentation	++	0
3 - Stream Source Connector	Extent	++	++
	Simplicity	++	++
	Documentation	++	+
4 - Preprocessing	Extent	++	++
	Simplicity	++	++
	Documentation	++	+
5 - Stream Processing	Extent	+	++
	Simplicity	+	++
	Documentation	++	++
Basic functions:		24	20
6a - Time Series Analysis - Adaptation to stock market use case	Adaptability	++	+
6b - Time Series Analysis - Integrating error calculation	Adaptability	++	++
7 - Evaluation	Adaptability	+	0
8 - Visualization	Adaptability	++	-
Advanced functions:		7	2
$\sum$	Total	31	22

Tab. 4: A qualitative comparison of a predictive analytics task between Flink and Spark.

Regarding the evaluation step, the functionality offered by Spark is less straightforward than the functionality offered by Flink. Visualization using the widely adopted tool, Grafana, can be easily integrated using Flink. Based on our analysis, it appears that integrating Flink with

Grafana for time series analysis is the preferred option for stream processing, monitoring, and the visualization of data streams. In Spark, there is no easy connector yet available for Grafana, despite the broad range of adopted tools. The focus of Spark does not lie on its streaming functionality, but more on batch. Flink's focus on streaming is remarkable in the majority of the tasks tested within this work and therewith brings qualitative higher *Ease of development*.

5 Acknowledgments

This work was supported by the German Federal Ministry of Economics and Technology (BMWi) funded *SePiA.Pro* project, under grant *FKZ: 01MD16013*.

References

[Ar01] Armstrong, J. Scott: Selecting Forecasting Methods. In: Principles of Forecasting, volume 30, pp. 365–386. Springer US, 2001.

[BKF17] Bader, A.; Kopp, O.; Falkenthal, M.: Survey and Comparison of Open Source Time Series Databases. In (Mitschang, Bernhard et al., eds): Datenbanksysteme für Business, Technologie und Web (BTW2017) – Workshopband. volume P-266 of Lecture Notes in Informatics (LNI). Gesellschaft für Informatik e.V. (GI), pp. 249–268, 2017.

[Ch16] Chintapalli, S.; Dagit, D.; Evans, B.; Farivar, R.; Graves, T.; Holderbaugh, M.; Liu, Z.; Nusbaum, K.; Patil, K.; Peng, B. J.; Poulosky, P.: Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming. IEEE, pp. 1789–1792, 2016.

[JPT17] Jensen, S. K.; Pedersen, T. B.; Thomsen, C.: Time Series Management Systems: A Survey. IEEE Trans. on Knowledge and Data Engineering, 29(11):2581–2600, November 2017.

[Ka18] Karimov, J.; Rabl, T.; Katsifodimos, A.; Samarev, R.; Heiskanen, H.; Markl, V.: Benchmarking Distributed Stream Data Processing Systems. CoRR, abs/1802.08496:12, 2018.

[KKR15] Kejariwal, A.; Kulkarni, S.; Ramasamy, K.: Real Time Analytics: Algorithms and Systems. Proc. VLDB Endow., 8(12):2040–2041, August 2015.

[SG17] Saxena, S.; Gupta, S.: Practical real-time data processing and analytics: distributed computing and event processing using Apache Spark, Flink, Storm, and Kafka. Packt, 2017. OCLC: 1008968663.

[WB01] Welch, G.; Bishop, G.: An Introduction to the Kalman Filter. Technical report, 08 2001.

[Wi16] Wingerath, W.; Gessert, F.; Friedrich, S.; Ritter, N.: Real-time stream processing for Big Data. deGruyter, (58), August 2016.