Poznań University of Technology

Faculty of Computing

Comparison technology of data streams analysis (Kafka Streams, Spark Streaming, Storm)

Autors:
Kamila Ziemba,
Joanna Chromińska,
Wiktoria Leitgeber
Bartosz Markowski

Supervisor:
Assoc. Prof., PhD DSc
Robert Wrembel

Poznań, September 19, 2017

Contents

1 Introduction **2**

2 System architecture **3**

2.1 The stream generator 3

2.2 Kafka 3

2.3 Kafka Streams 4

2.4 Spark Streaming 5

2.5 Storm 5

3 Criteria for data analysis **7**

3.1 Efficiency 7

3.2 Functionality 7

3.2.1 Pattern detection 7

3.2.2 Aggregation 9

4 Testing environment **10**

5 Comparison of Kafka Storm and Spark technology **15**

6 Conclusions **16**

1 Introduction

The purpose of the project is to compare real-time data stream analysis technology such as Kafka Streams, Spark Stream and Storm. The real-time processing and analysis becomes a key element for most organizations. Streaming data is a group of data records generated from a variety of sources, such as sensors, server traffic and online searches.

In this project, the data source is a stream generator that generates random, synthetic data in real-time at specified intervals. Comparison of Spark and Storm is based on the speed of data retrieval and processing at different data stream frequencies. The second criterion of analysis is system functionality: data aggregation, sum and average counting, minimum and maximum values detection and pattern detection.

2 System architecture

The system architecture that is used in the project (Figure 1) includes a data stream generator which generates synthetic, floating point data that is sent to the Kafka. Three systems (Kafka Streams, Spark Straming, and Storm) retrieve data from the Kafka and then performs various operations on it.

This is n-node architecture, unfortunately, because of the lack of available ports, it was not possible to perform tests in a lab room on n computers.

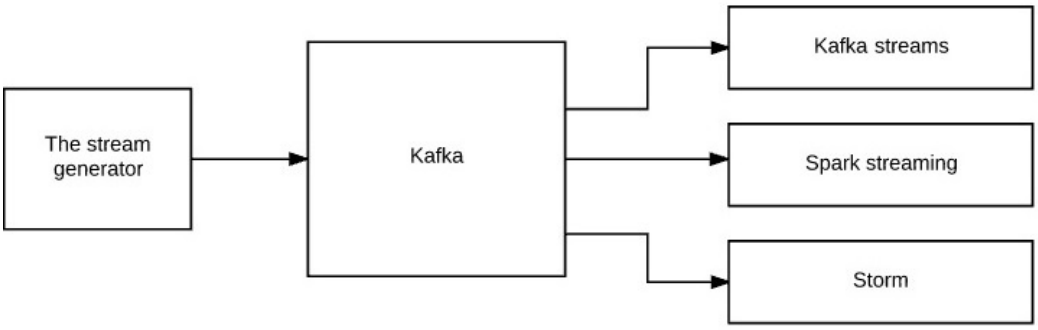

Figure 1: Graphical presentation of system architecture

2.1 The stream generator

The data stream generator is used to generate synthetic, floating point data at specified intervals. This is a simulation of energy data. This makes it possible to control the frequency of data generation. Kafka retrieves data from the source, bufforing them, and inserts it into the queue.The next step is the analysis of data streams by three independent modules: Kafka Streams, Spark Streaming and Storm. This component is implemented using Java (Eclipse Java Neon environment) and the Kafka Producer Api.

2.2 Kafka

The Kafka was created in order to send messages between many applications and it is centred on distributed streaming. The Kafka's communica-

tion it is grouped into „Topics" to which are messages sent by group called „Producers". They can be for example some sensor. During this study this mechanism was used in order to send dynamically generated stream to so-called „Consumers" - the programms that read data from stream and they analyse it on, for instance, „Windows" which have some certain lenght. The Kafka characteristic is that it has a really great efficiency so it's used in order to analyse great amount of stream data. In order to manage the distributed streaming condition, Kafka uses „Zookeper".

2.3 Kafka Streams

Kafka Streams, a component of open source Apache Kafka, is a powerful, easy-to-use, client library for building highly scalable, fault-tolerant, distributed stream processing applications on top of Apache Kafka. It builds upon important concepts for stream processing such as properly distinguishing between event-time and processing-time, handling of late-arriving data, and efficient management of application state.

Kafka Streams builds streaming applications, specifically applications that transform input Kafka topics into output Kafka topics (or calls to external services, updates to databases, etc.). It lets make concise code in a way that is distributed and fault-tolerant. Stream processing is a computer programming paradigm, equivalent to data-flow programming, event stream processing, and reactive programming, that allows some applications to more easily exploit a limited form of parallel processing.

Kafka Streams simplifies streaming applications is that it fully integrates the concepts of tables and streams. A stream is the most important abstraction provided by Kafka Streams: it represents an unbounded, continuously updating data set, where unbounded means "of unknown or of unlimited size". Stream consists of one or more stream partitions. A stream partition is an sequence of immutable data records, where a data record is defined as a key-value pair. That sequence is ordered, replayable, and fault-tolerant. Tables are particular view on a stream, a cache of the latest value for each key in a stream.

2.4 Spark Streaming

Spark Streaming is an extension of core Spark API. Apache Spark is a data parallel general purpose batch processing engine. Spark Streaming makes it easy to build fault-tolerant processing of real-time data streams. The way Spark Streaming works is it divides the live stream of data into batches of a pre-defined interval (N seconds) and then treats each batch of data as Resilient Distributed Datasets (RDDs). The results of these RDD operations are returned in batches. In implemented batches we can aggregate and analysis data. It's important to decide the time interval for Spark Streaming, based on your use case and data processing requirements. It the value of N is too high, then the batches will have enough data to give meaningful results during the analysis.

Streaming data can come from many different sources. In these project data sources include the Apache Kafka and Kafka Streaming, which generate random real-time data streams in specific time. For example generate one data in one second or faster/slower. The module was implemented in Java language in Eclipse Java Neon environment on the Windows 10 operating system. Program is processing in real-time data streams into batches in predefined interval, and analysis data streams in design pattern - if pattern will occur, the program shows the relevant information.

2.5 Storm

Apache Storm is a free and open source distributed realtime computation system for processing large volumes of high-velocity data. Storm is simple and can be used with any programming language. Storm has five characteristics which makes Storm ideal for real- data processing workloads. Storm is fast, scalable, foult-tolerant, reliable and easy to operate. Storm has many use cases: realtime analytics, online machine learning, continuous computation and more. There are several different components in the Storm archite/cture, such as:

- tuple - the main data structure. It is a list of elements.
- stream - is an unorderd sequence of tuples.

- spouts - is the soutrce of stream. Spout reads data from datasource like Apache Kafka.
- bolts - are logical processing units. Spouts pass data to bolts and bolts process tumples and produce a new tumples as output. Bolts can perform the operations of filtering, aggregation and joining. Bolt receives data and emits to one or more bolts. Bolt implements the IRichBolt interface.

In the described project Java language and Eclipse Java Neon environment was used to implementation. The program uses two "bolts" and one "spout" components. The spout component is used for connection to the Kafka. The first bolt component implements the calculation of the average, sum and maximal values from the window of the given length. The second bolt component implements pattern (W) detection in a five element windows. If the program detects a pattern, the program will display the appropriate message.

3 Criteria for data analysis

This chapter deals with comparative analysis of two systems: Spark Streaming and Apache Storm. Within comparative analysis, a performance and functional test were performed. The analysis concerns the processing speed of data at different data stream frequencies. Comparison of the functionality of both systems deal with the aggregation capabilities and correctness of pattern detection in the data stream.

3.1 Efficiency

The evaluation and comparison of technology in terms of efficiency was based on data stream intensity analysis. Efficiency measure is the amount of message the program is able to read and process in one second.

The performance test involves generating a data stream of varying intensity. Spark and Storm systems retrieve data streams from Kafka during this time. The processing speed of both systems is examined by the length of the data stream and the location (offset) of the data processing program. Increasing the intensity of the stream is interrupted when the data processing programs are not keeping up with the data retrieval or when the workstation was no longer able to generate data faster (a co-processor power limitation). The processing speed measurement was tested for calculation of the sum, average and for pattern detection.

3.2 Functionality

Spark and Storm functional analysis concerns the pattern detection, calculating sum, average and minimal and maximal value in a stream. Functionality is also the ability to aggregate data from the stream for further processing.

3.2.1 Pattern detection

Pattern detection involves detecting singular values in a data stream that deviate from a defined data model. In the project we used pattern

referred to the model (1):

$$A > B < C > D < E \tag{1}$$

Testing the pattern detection speed in the data stream is analogous to the performance test. The stream is generated with varying intensity, in the meantime, Spark and Storm retrieve data and detect patterns. Processing speed is calculated based on the stream length and program position.

Figure 2 shows a graphical representation of a defined outlier detection model. It is compatible with equation (1). Figure 3 presents the reaction to detect outlier values in a stram.

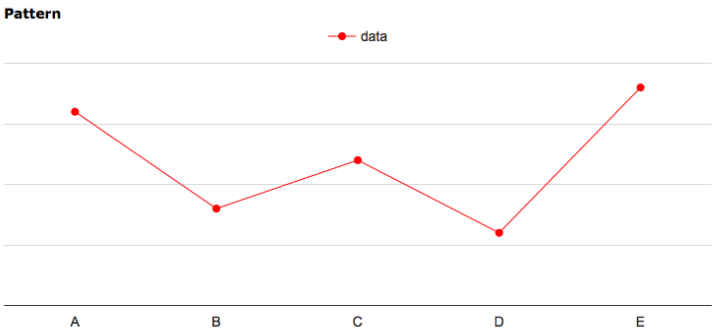

Figure 2: Example pattern detection for Model 1

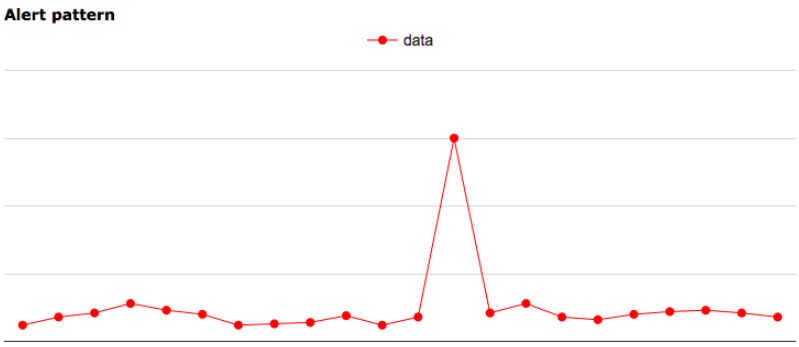

Figure 3: Reaction to an abnormal event

The pattern is designed to respond to events with alarms of the detection distant values.

3.2.2 Aggregation

In both technologies you can create both a time window (for example 10 minutes) and a quantitative window (for example 100 elements). On the aggregated data in the window operations are performed for calculating the sum, average, and minimum and maximum values. Aggregation tests have been successful in all systems, differences between them are minimal.There was a slight delay caused by system overload by other applications, but it did not affect the correctness of aggregation and read data from the stream.

Several different aggregation windows were used to implement all operations in Storm program. Both quantitative and time windows were used.

4 Testing environment

The tests of each of the systems were performed on workstation with windows 10 and with the following technical specifications: I5-6300m 8gb ddr4 2400 mz plus 512 ssd. All programs use the Java version 8.

Kafka Offset Monitor was used for data analysis. This is an app to monitor kafka consumers and their position (offset) in the stream. We can see the current consumer groups, for each group the topics that they are consuming and the position of the consumer in each topic. This is useful to understand how quick are consuming from a queue and how fast the queue is growing.

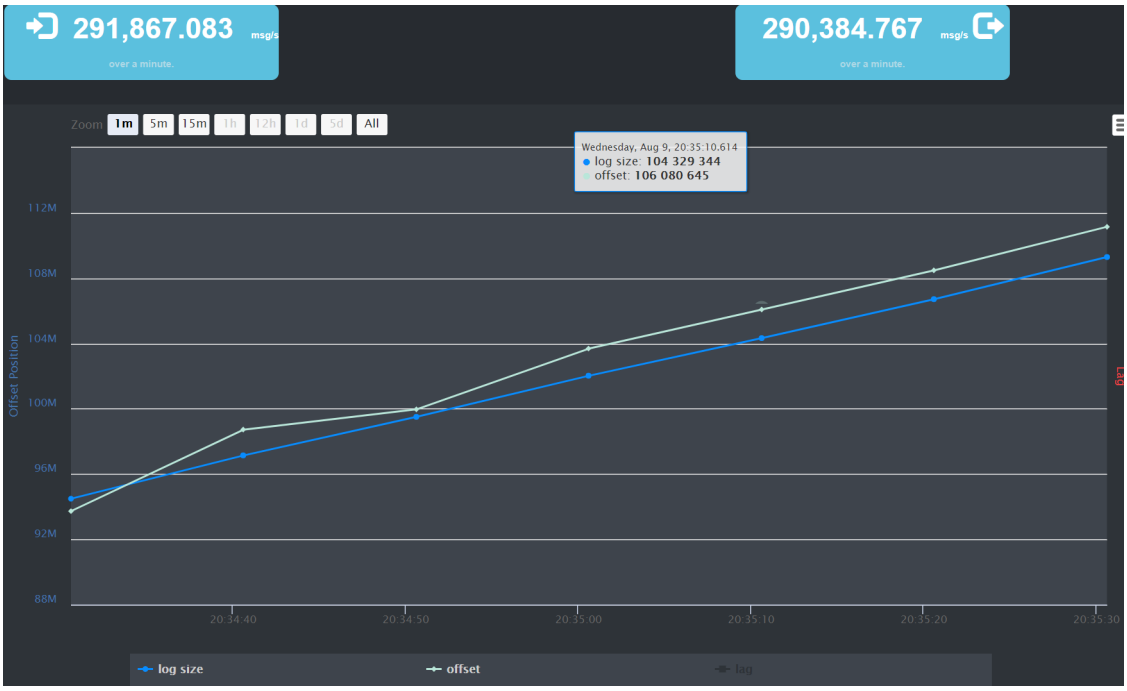

Figure 4: Spark

Figure 3 presents a screenshot from the stream analysis program. The log size (blue) shows the total length of the stream, offset (white) shows which element is consuming. Because all calculations are on one computer, the results are a minor error. Offset reading is done after the queue length reading, which at this load introduces a slight delay and causes the offset to

be larger. If we were to read it exactly at the same time as the length of the queue, the value would be less. On the test computer, we managed to achieve a speed of 290,000 messages per second. This is the maximum speed with which computer was able to generate and receive data. We present one image because the same result was obtained for calculating mean, sum and counting elements. Our resources did not allow us to fully test Spark. The speed of data transfer was too fast, so it could not be precisely tested without separate servers.

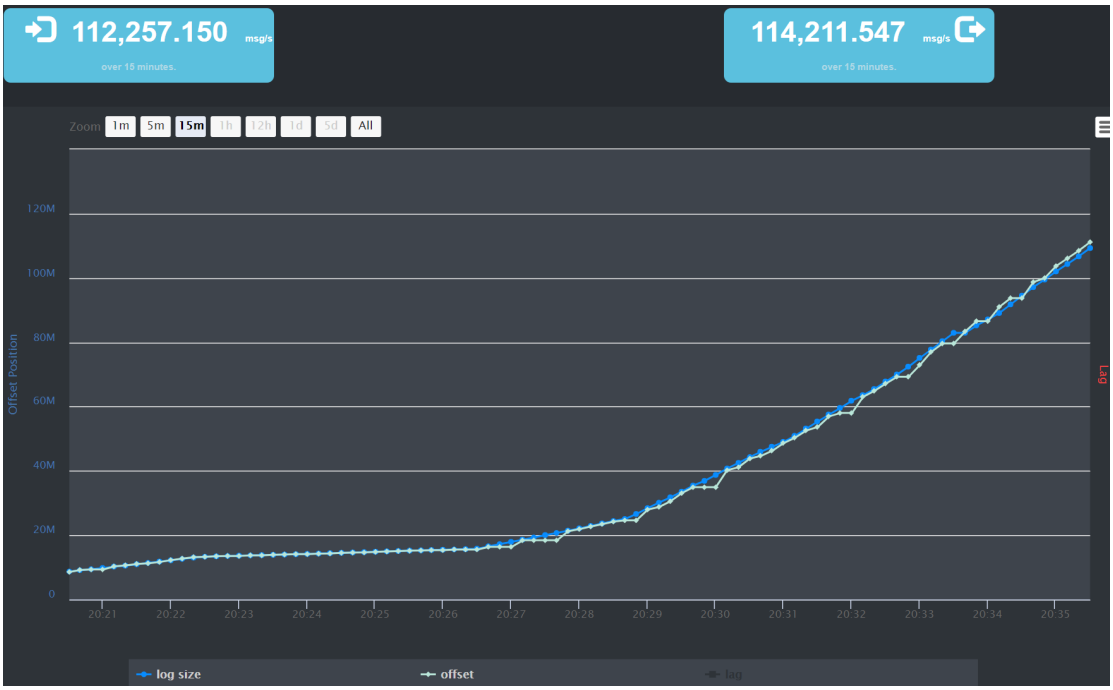

Figure 5: Storm count message

Figure 4 shows how the Storm tests ran. This test was about element counting. In the beginning, the input stream has a small speed, with time increases. At a speed of about 145,000 messages per second we reached the maximum stream velocity that Storm was able to process.

Another test was to count the sum in the window. The final result is shown in figure 5. This graph presents the delay (red).

In this case, we also significantly increased the data generation rate to 210K per second. From the graph and statistics above it shows that Storm in processing the sum processed 112k messages per second.

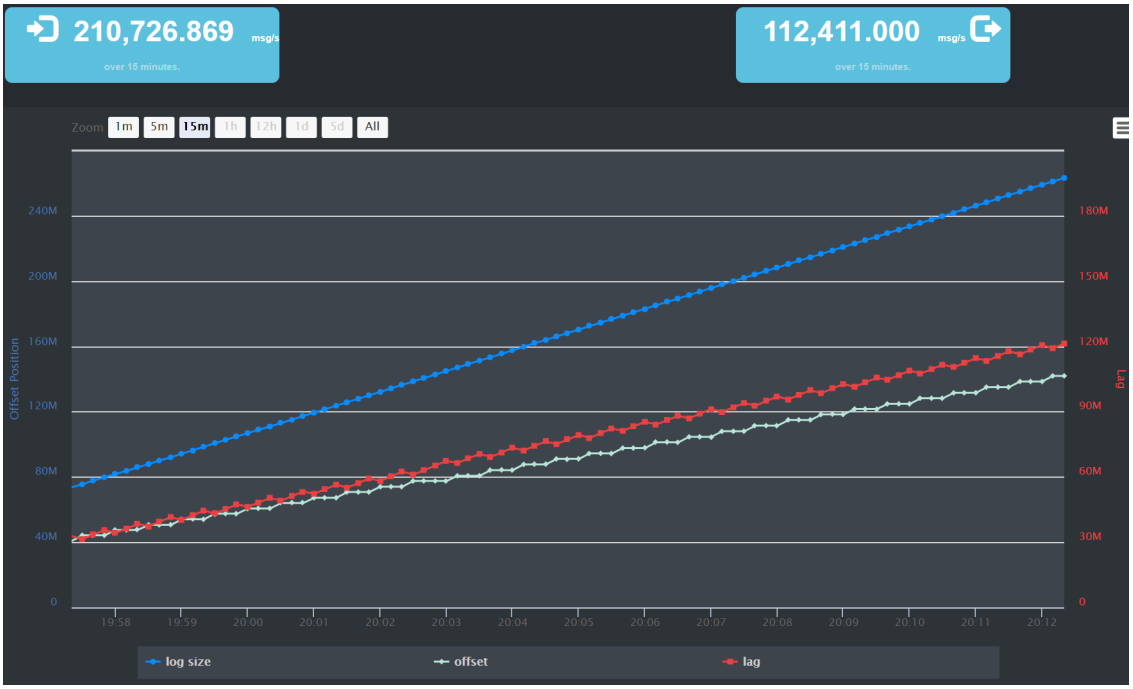

Figure 6: Storm sum of element value

Figure 6 shows the effects we have achieved when calculating the average using Storm. We got a speed of 97k messages per second. Calculating the average was the most computationally intensive and yielded the weakest result.

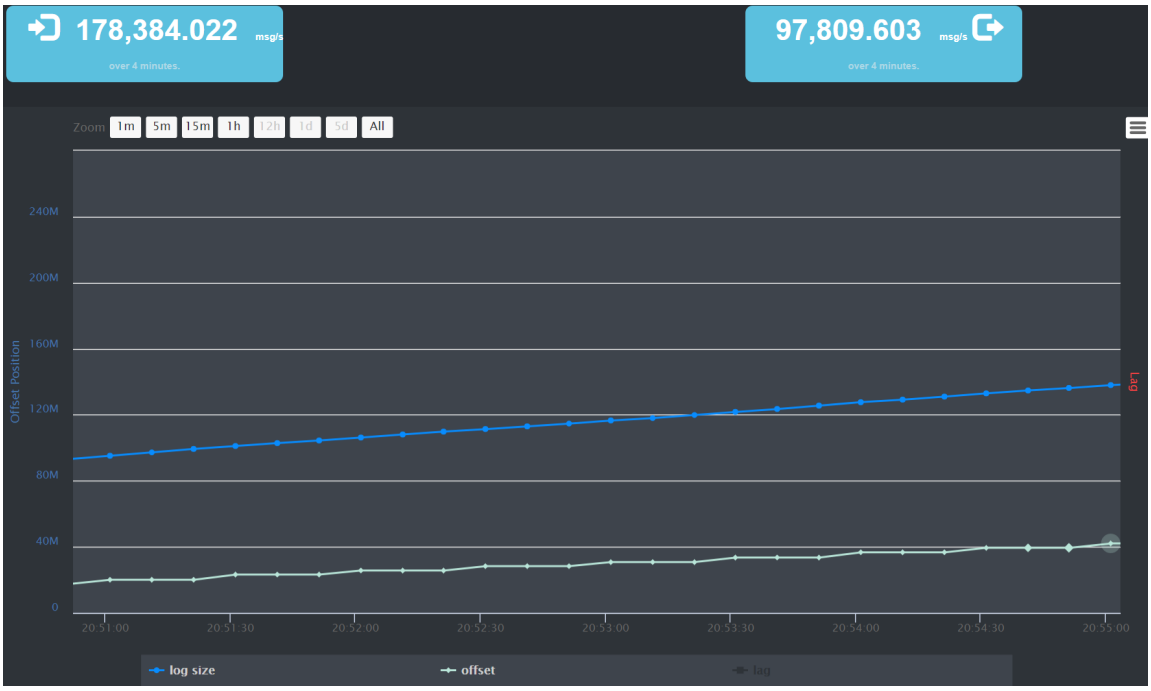

Figure 7: Storm average

Unfortunately, speed measurement in our case does not work - the application hangs up (exception error) after a long time. Therefore speed is not equivocal. In a normal environment, pattern detection works fine. The correctness of the pattern detection was tested on the test data, which was shown during the consultation.

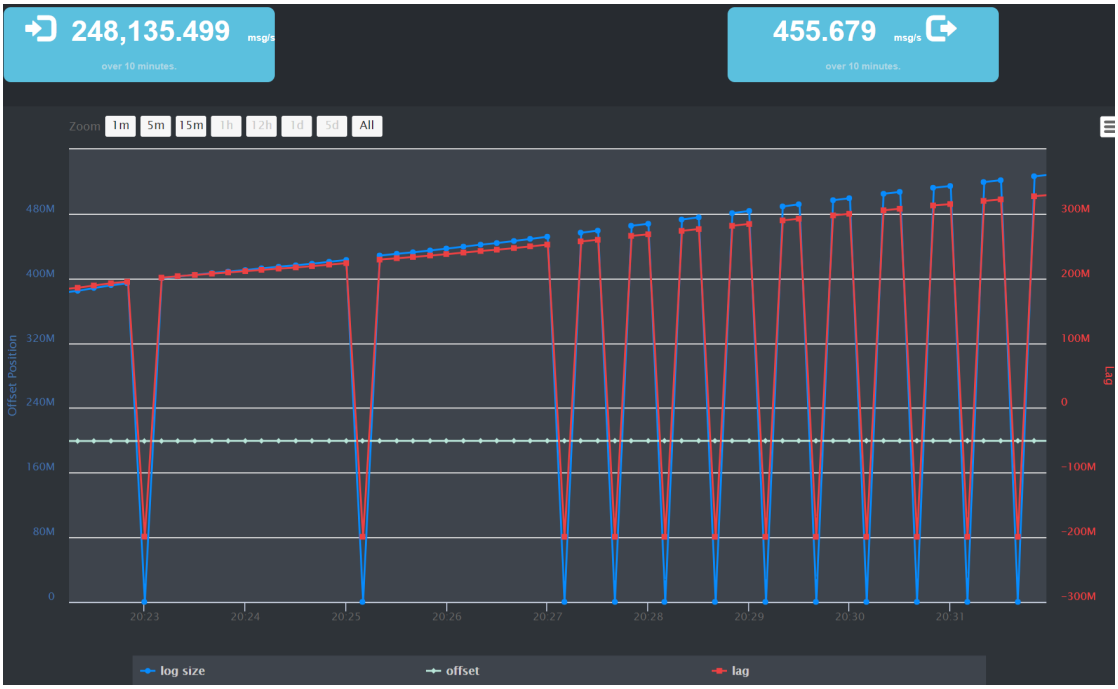

Figure 8: Kafka crash

For the Kafka Streams, there were huge memory errors that made it impossible to test performance measurement with Kafka Offset Monitor. There was a configuration issue of the environment used. This incorrect behavior can be seen in figure 7. In addition, the application was able to turn off.

5 Comparison of Kafka Storm and Spark technology

The graph below shows the average speed for the spark and storm system. The spark system is much more efficient and faster than storm system. Kafka Streams module was not included due to lack of possibility to measure speed.

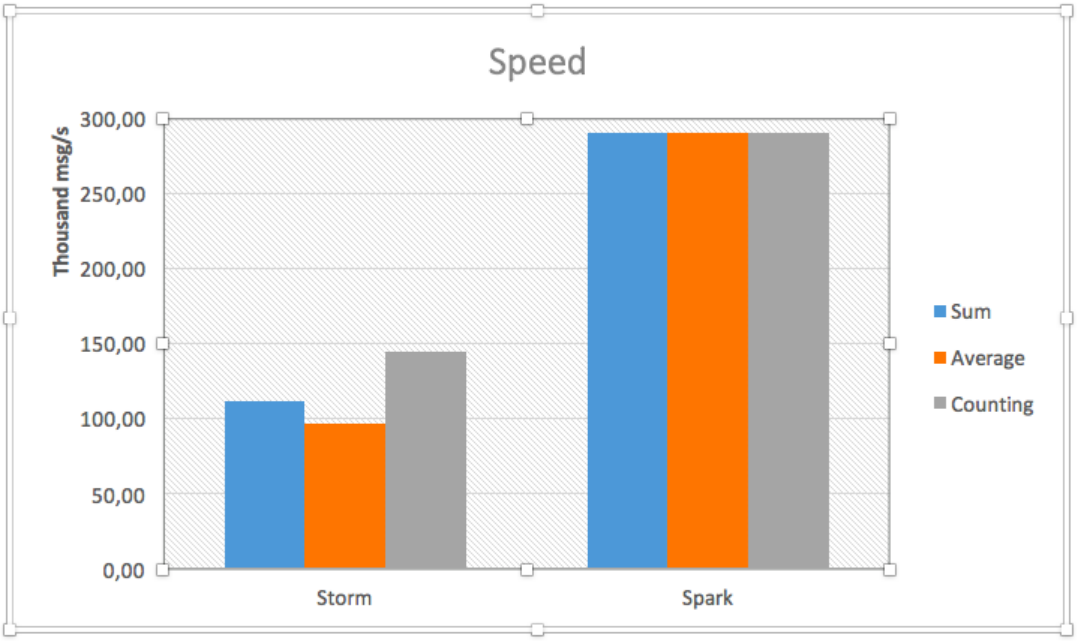

Figure 9: Comparison of maximum speeds for Storm and Spark.

As described in the testing section, we did not measure the exact pattern detection speed.

The speeds for detecting patterns in both systems were comparable.

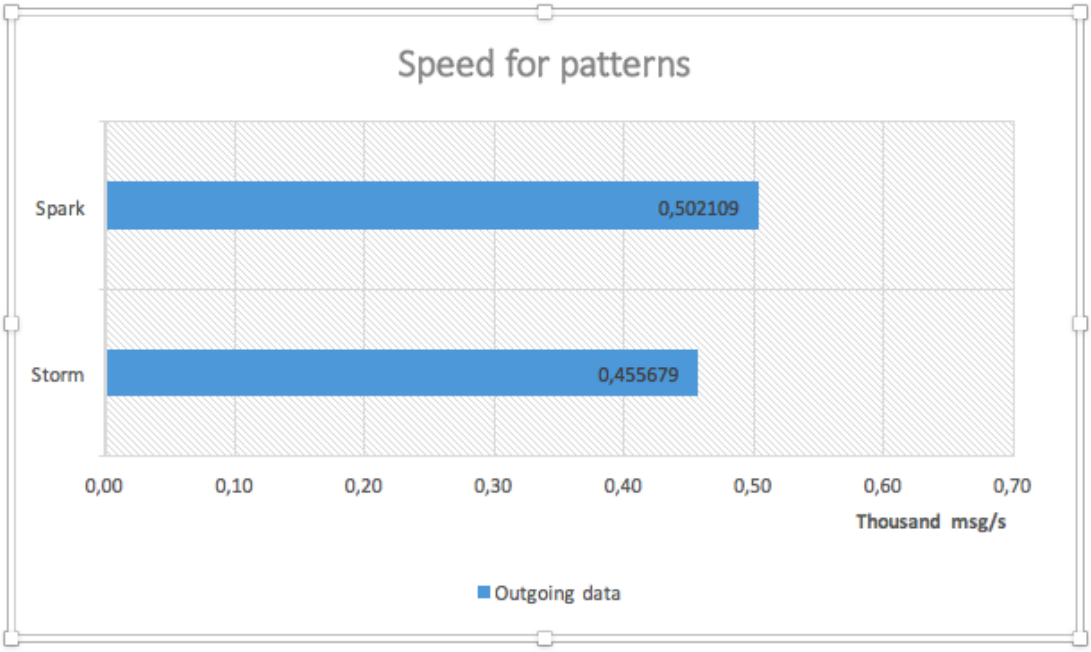

Figure 10: Comparison of Spark and Storm for pattern detection.

6 Conclusions

The experiments show that for the criteria we set the best is the Spark system. It guarantees good processing speed, functionality and reliability. Storm is also very good, although due to the occurrence of minor delays it is less efficient than Spark.

There was a problem with integration Spark system and Apache Kafka in python language, so instead of python was used java for implementation. In this respect, the Storm system was better.

The worst system in our project is Kafka Streams. There were problems with the environmental integration and the latest implementation. Experienced huge memory errors that made it impossible to test performance measurement with Kafka Offset Monitor. There was a configuration issue of the environment used.

Although this is n-node architecture, tests on n nodes were not per-

formed due to the lack of available ports on workstations in the lab.

References

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

[2] https://www.infoq.com/articles/apache-spark-streaming

[3] http://docs.confluent.io/current/streams/developer-guide.html