


Big Data Streaming Platforms to Support Real-time Analytics

Eliana Fernandes¹, Ana Carolina Salgado²^a and Jorge Bernardino^{1,3}^b

¹*Polytechnic of Coimbra – ISEC, Rua Pedro Nunes, Quinta da Nora, 3030-199 Coimbra, Portugal*

²*Centre for Informatics, Universidade Federal de Pernambuco, Recife, Brazil*

³*Centre for Informatics and Systems of the University of Coimbra (CISUC), Portugal*

Keywords: Streaming, Real-time Analytics, Big Data, Fault-Tolerance.

Abstract: In recent years data has grown exponentially due to the evolution of technology. The data flow circulates in a very fast and continuous way, so it must be processed in real time. Therefore, several big data streaming platforms have emerged for processing large amounts of data. Nowadays, companies have difficulties in choosing the platform that best suits their needs. In addition, the information about the platforms is scattered and sometimes omitted, making it difficult for the company to choose the right platform. This work focuses on helping companies or organizations to choose a big data streaming platform to analyze and process their data flow. We provide a description of the most popular platforms, such as: Apache Flink, Apache Kafka, Apache Samza, Apache Spark and Apache Storm. To strengthen the knowledge about these platforms, we also approached their architectures, advantages and limitations. Finally, a comparison among big data streaming platforms will be provided, using as attributes the characteristics that companies usually most need.

1 INTRODUCTION

The explosive growth of the Internet has caused large amounts of data to be generated. The companies try to react to this evolution and if data isn't processed efficiently and at the same speeds (Safaei, 2017).

Big data is a generic term for organizing, processing, and aggregating large amounts of data. The data that has a fast and continuous changing is called streaming data (Behera *et al.*, 2018). It needs to be analyzed in a short period of time. Traditional Business Intelligence tools aren't suitable for analyzing streaming data in real time, because is processed in batch processing (Behera *et al.*, 2018). A large number of big data streaming platforms have been developed (Imanuel, 2019).

Big data streaming platforms are the main challenge for most companies. The requirements of companies are sometimes different from the features that these platforms offer. The objective of this work is to assist in choosing a big data streaming platform, taking into account the characteristics that platforms may have for companies. As well as, is to describe and compare the most popular and open-source big

data streaming platforms, such as: Flink, Kafka, Samza, Spark and Storm (Imanuel, 2019).

The rest of this paper is structured as follows. Section 2 provides an overview of the big data streaming platforms, their architecture, advantages and limitations. Section 3 presents a comparative study of these platforms. The conclusions and future work are presented in Section 4.


2 STREAMING PLATFORMS


Processing data means manipulating, aggregating in order to transform data into useful information.

Big data streaming processing is always up-to-date. So, when the data is available, it's processed immediately and is transformed into information.

To ensure continuous and stable operation of the entire system it is necessary that the platform has a suitable architecture design. The architectures for big data streaming platforms, can be: symmetrical architecture and master-slave architecture.

In symmetrical architecture, the functions of each node are the same and have good scalability.

^a <https://orcid.org/0000-0003-4036-8064>

^b <https://orcid.org/0000-0001-9660-2011>

However, as there is no central node, the system must contain resource scheduling, system fault tolerance and data balancing (Sun *et al.*, 2019).

The master-slave architecture has one master node and several slave nodes. The master node manages system resources, coordinating tasks, completing system fault tolerance and balancing data. The slave node receives tasks from the master node. Throughout the process, there is no data exchange between slave nodes, and system-wide operations are completely dependent on master node control (Sun *et al.*, 2019).

To solve the problem of large-scale real-time processing, current big data streaming platforms, such as Flink, Kafka, Samza, Spark Streaming, and Storm, have emerged. These platforms adopt the master-slave architecture.

There are a lot of big data streaming platforms. However, many of them are only used for batch processing, such as Hadoop. The chosen platforms can handle data in real time and can perform streaming processing. Another important issue for the choice of platforms was the availability to use, that is why we have analyzed only open-source platforms. In addition, these platforms have a huge community of developers and users (Neves and Bernardino, 2015).

Finally, the selection of the platforms was also given to the popularity of a platform itself, the wealth of resources and its usefulness. We took into account some characteristics, such as ease of use, number of features, among others (Immanuel, 2019).

2.1 Apache Flink (flink.apache.org)

Flink is an open source platform for distributed stream and batch data processing (Stratosphere and Markl, 2018). It's a platform that provides data distribution, and fault-tolerance for data stream calculations (Stratosphere and Markl, 2018). Its processes the user-defined functions code through the system stack. It's ability to compute common operations (Nasiri, Nahesi and Goudarzi, 2019).

2.1.1 Flink Architecture

The platform offers software developers various application programming interfaces (APIs), for creating new applications to be executed on the Flink engine. Examples of these APIs, represented in Figure 1 (Stratosphere and Markl, 2018).

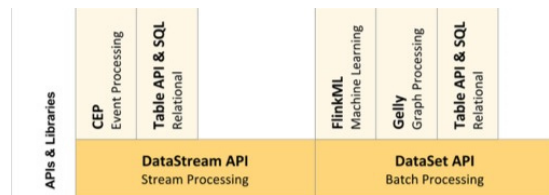


Figure 1: APIs and Libraries of Flink (Shahverdi, 2018).

The main Flink APIs are the Batch DataSet API and the Streaming DataStream API. In this case we will approach the DataStream API, because this API allows to handle a large amount of data in real time. This API performs filtering, updates, window definition, joins etc. It can receive any kind of data from message queues, sockets, and file systems (Shahverdi, 2018).

Flink stream processing model handles incoming data on an item-by-item basis as a true stream. Flink provides its DataStream API to work with unbounded streams of data. The basic components that Flink works with are (Gurusamy, Kannan and Nandhini, 2017):

- Streams are immutable, unbounded datasets that flow through the system;
- Operators are functions that operate on data streams to produce other streams;
- Sources are the entry point for streams entering the system;
- Sinks are the place where streams flow out of the Flink system. They might represent a database or a connector to another system.

The APIs present a logical representation and are converted to a directed acyclic task graph that is sent to the cluster for execution. A Flink cluster, shown in Figure 2, comprises three types of processes: the client, the job manager, and at least one task manager.

The client takes the program code, transforms it to a dataflow graph, and submits to the job manager. This transformation phase also examines the data types of the data exchanged between operators and creates serializers and other type/schema specific code (Katsifodimos and Schelter, 2016).

Job manager coordinates distributed execution of the data stream. It tracks the status and progress of each operator and flow, schedules new operators, and coordinates checkpoints and recovery points.

Actual data processing takes place in task managers. And it runs one or more operators that produce streams and reports their status to the task manager. Job managers maintain buffer pools for buffering or materializing streams and network connections to exchange data streams between operators (Katsifodimos and Schelter, 2016).

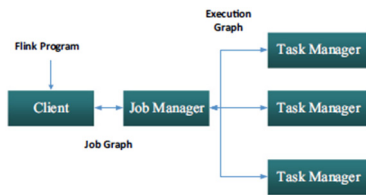


Figure 2: Flink Processing Flow (Nasiri, Nahesi and Goudarzi, 2019).

2.1.2 Flink Advantages and Limitations

Flink have some advantages, such as (Levy, 2019):

- It doesn't require manual optimization and adjustment to data it processes;
- Dynamically analyzes and optimizes tasks.

Flink has also some shortcomings (Sun *et al.*, 2019):

- With a large number of changes to external events, it cannot know how large-scale resources are needed, creating workload issues;
- Some scaling limitations.

2.2 Apache Kafka (kafka.apache.org)

Kafka is a highly available open source, fault-tolerant, scalable distributed streaming platform. It can be used to store and process data streams, and is intended to provide unified, high throughput, low latency platform for handling feeds of real time data (Freiknecht *et al.*, 2018). Kafka was introduced by LinkedIn in 2011 and is written in Scala and Java (Shaheen, 2017). Kafka is a publishing and subscribing messaging system. A Messaging System is responsible for transferring data from one application to another and focus on data. Distributed messaging is based on the concept of reliable message queuing. There are two types of messaging patterns available (Team, 2019):

- Point to Point Messaging System – messages remain in a queue. More than one consumer can consume the messages in the queue;
- Publish-Subscribe Messaging System – messages remain in a topic. Consumers can take more than one topic and consume every message in that topic.

2.2.1 Kafka Architecture

Kafka is deployed as a cluster on multiple servers, so it handles its entire publish and subscribe messaging system with the help of four APIs, such as: producer, consumer, streams processors and connector.

- Producer API: customers can connect to Kafka servers, and customers can post the log stream to one or more Kafka topics.

- Consumer API: Allows clients to connect to Kafka servers running in the cluster and consume streams of records from one or more Kafka topics. This platform consumes the messages from Kafka topics.
- Streams API: Clients act as flow processors by consuming flows from one or more topics and producing flows to other output topics. This allows to transform input and output streams.
- Connector API: Allows writing reusable producer and consumer code. We can create reusable source and sink connector components for various data sources.

Figure 3 shows a short illustration of the Kafka ecosystem. It shows how producers send messages to the cluster and presents how consumers extract this message from the broker. Also, it can see the Zookeeper, which is used to manage and coordinate the Kafka cluster. The Zookeeper is used to notify producer and consumer of the presence of any new broker in the system or broker failure (Shaheen, 2017).

A Kafka cluster is made up of connectors that record changes to records in a relational database, data producers, data consumers or data processors (TutorialKart, 2019). The main components of its architecture are topics, registers and intermediaries.

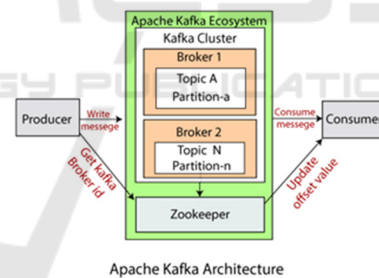


Figure 3: Kafka Ecosystem (JavaTpoint, 2020).

Topics consist of a flow of records containing different information (Shaheen, 2017). Data or messages are partitioned into different partitions within various topics. Here, messages are indexed and stored associated with a data / time stamp. Consumers can consult messages from these parties.

2.2.2 Kafka Advantages and Limitations

Kafka are some of the advantages (Instaclustr, 2019):

- Load balance and data replication;
- Can handle high-velocity of data;

There are some limitations (JavaTpoint, 2019):

- The Kafka broker can sometimes have problems when a message needs some tuning as Kafka's performance is reduced;
- Brokers and consumers reduce Kafka performance by compressing and decompressing data flow, thus affecting performance and throughput;

2.3 Apache Samza (samza.apache.org)

Samza was developed by LinkedIn. Is a distributed flow processing platform and also an open source Kafka message queue-based system for implementing real-time flow data processing (Sun *et al.*, 2019). It is formed by combining Kafka and YARN to perform the computation of data streams (Nasiri, Nahesi and Goudarzi, 2019). Samza is designed to take advantage of Kafka's unique architecture and warranties, although Kafka can be used by other flow processing systems. That's why Samza uses Kafka to provide fault-tolerance and state (Gurusamy, Kannan and Nandhini, 2017).

This platform supports a high throughput for a wide variety of processing standards while providing operational robustness at the massive scale required. To achieve its primary purpose it uses a small number of carefully designed abstractions partitioned message logs, fault-tolerance local state, and cluster-based scheduling (Kleppmann, 2018). The core of Samza consists of several fairly low-level abstractions, on top of which high-level operators have been built.

2.3.1 Samza Architecture

The Samza architecture consists of the flow data layer (Kafka), the execution layer (YARN), and the processing layer (Samza API) (Sun *et al.*, 2019). It is used for consuming flows, processing messages, and producing derived output streams. One of Samza work consists of a Kafka consumer, an event loop that calls the application code to process incoming messages, and a Kafka producer that sends outgoing messages back to Kafka. YARN is used to automatically restart failed processes, metrics, and monitoring. It even plays the role of resource manager and cluster manager. For processing messages, Samza provides a Java StreamTask interface that is implemented (Kleppmann and Kreps, 2015).

A node manager demon is running at each node in the cluster and is responsible for scheduling the process on the node (Behera *et al.*, 2018). A resource manager is responsible for coordinating the task executed at each node in the cluster. Work progress

or resource failure at slave node is reported periodically by the node manager. Node managers might communicate among themselves. Resource manager and Node manager are communicated by a concept known as "heartbeat" (Behera *et al.*, 2018).

Kafka works at the streaming layer, and acts as a distributed Message Queuing system that provides at least once the message delivery guarantee policy. Each data stream is known as a topic that is partitioned and replicated across multiple nodes. When a producer sends a message to a topic, a key is provided and determines the partition to which the message is to be sent (Behera *et al.*, 2018).

Kafka's provides Samza with some features that are difficult or should not be implemented in other streaming platforms. The Kafka cluster consists of several intermediate servers. On this, each message type is defined as a topic. Messages on the same topic are partitioned and stored in different intermediaries, according to a given key and algorithm.

2.3.2 Samza Advantages and Limitations

Samza have some advantages, such as (Levy, 2019):

- Provides reliable persistence with low latency, offering replicated storage;
- Can eliminate backpressure, allowing data to be persisted and processed later.

Although Samza has many advantages, it also has some limitations, such as (Sun *et al.*, 2019):

- There is no full fault-tolerance, causing state information in the memory of the source node to be lost when the node fails to transfer;
- Only supports JVM languages;
- Doesn't support very low latency.

2.4 Apache Spark (spark.apache.org)

Spark is an open source big data streaming platform, developed in 2009 by Matei Zaharia (Vaidya, 2019). It's designed to support iterative algorithms, interactive queries and streaming. And it's highly scalable, high fault-tolerance, high performance and low latency (*Apache Spark - Introduction*, 2019). Spark allows for ease of developing large-scale applications, and it has some scalability issues (Ghasemi and Chow, 2019). This system supports various programming languages, such as Java, Python, Scala (Behera *et al.*, 2018). This platform provides large number of tools, as shown in Figure 4, for example, stream processing engine called Spark Streaming (Shoro and Soomro, 2015).

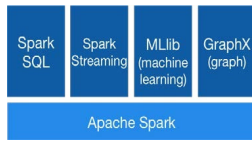


Figure 4: Tools of Spark (Shoro and Soomro, 2015).

Spark can handle real-time data with an extension feature called Spark Streaming. The advantage of using Spark Streaming is that it can handle both batch and streaming data. It also helps Spark to increase its primary scheduling capability and perform streaming analysis on real-time data.

2.4.1 Spark Architecture

Spark has a well-defined layered architecture integrated with many extensions (Vaidya, 2019). The architecture of Spark is illustrated in Figure 5:

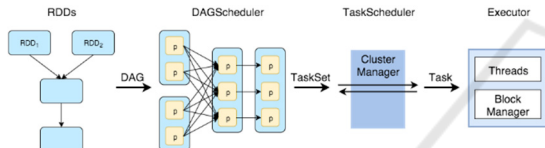


Figure 5: Structure of Apache Spark (Kirillov, 2016).

- *Resilient Distributed Dataset (RDD)* – is a partitioned collection of elements that can be operated in parallel. Each data set runs on different nodes of a cluster;
- *Directed Acyclic Graph (DAG) Scheduler* - is a graph that is directed and without cycles connecting the other edges. The edges of the directed graph only go one way.

Spark creates an operator chart, and when performing an action, the chart is sent to a DAG Scheduler. DAG Scheduler divides the graph into phases. A phase is made up of tasks based on partitions of the input data. At the end, the stages are sent to the Task Scheduler. The task scheduler starts tasks via the cluster manager.

In Figure 6 the cluster view of Spark is shown. In this cluster, the master ensures normal operation of the entire Spark system. The worker is the compute node, mainly used to accept the tasks of the master node (Sun *et al.*, 2019). System processing can be divided into three parts, including executor, cluster manager, and driver.

The master node converts the application into a set of tasks to be performed by a set of executors. It's then passed to cluster manager for distribution. The purpose of them is to distribute tasks to the most appropriate server in the cluster. Each server has an

executor who receives tasks from the cluster manager, executes them, and then returns the results (Nasiri, Nahesi and Goudarzi, 2019).

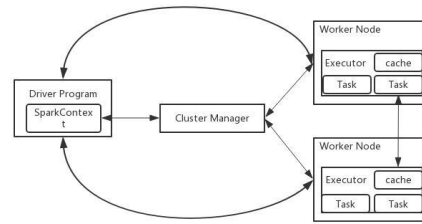


Figure 6: Architecture of Apache Spark (Sun *et al.*, 2019).

2.4.2 Spark Advantages and Limitations

There are many features of Spark that are very beneficial. So, there are several advantages of Spark:

- Efficient in iterative queries and immediate support in SQL queries (Levy, 2019);
- High-level Machine Learning Algorithms.

Although the Spark system provides solutions for streaming data on the time delay, fault-tolerance and throughput, there are also some unsolved shortcomings of the Spark system (Sun *et al.*, 2019):

- It can be complex to configure and deploy;
- The Spark system tends to be unstable and can only be used for calculations;
- Task scheduling efficiency is very low.

2.5 Apache Storm (storm.apache.org)

Storm is an open source big data streaming platform and can handle large amounts of data. Storm pioneered the wave of fault-tolerance distributed flow processing platforms (Shahverdi, Awad and Sakr, 2019). This platform was originally created by Nathan Marz in 2011 (Point, 2019).

Storm focuses on extremely low latency (Gurusamy, Kannan and Nandhini, 2017) and it's scalable, and easy to set up and operate (Foundation, 2019 b).

Storm has many use cases: real-time analytics, online machine learning, continuous computation, ETL (Extract, Transform, Load), and among others (Foundation, 2019 b). The Storm is written in Java and Clojure.

2.5.1 Storm Architecture

In Storm, the topologies are composed of multiple components that are arranged in a directed acyclic graph (DAG) of real-time computing. In a DAG the edges show us the data flow between them and the vertices show the components.

Storm topology consists of several components allowing to transfer one data stream to another stream in a reliable and distributive way. Storm data streams are precisely unlimited sequences of tuples, and also the data structure to represent standard data types or user-defined types with some additional serialization code (Hoseiny Farahabady *et al.*, 2016).

Spouts are the source of data streams. It allows a topology to retrieve data from external data generators for later transformation into standard tuples (Sun *et al.*, 2019). As a topology is fed by input tuples, Spouts can emit streams along the edges of the directed graph (Hoseiny Farahabady *et al.*, 2016).

Bolts are the processing nodes that receive Spout tuples, consume any number of input streams, perform some processing, and issue new streams (Shahverdi, Awad and Sakr, 2019). Bolts represent the logical components of the implementation of various flow processing operations.

In Storm, the process of a topology is always sent to the Zookeeper cluster. For running topologies, there are three types of entities (Shahverdi, 2018):

- Worker Process: it's processing executors within its topology. A topology can contain more than one worker process;
- Executor: This is a thread that was generated by the Worker Process. Executor processes perform tasks for Spouts and Bolts;
- Task: It's the entity that processes the data. In topology, multiple tasks are always equal or greater than the number of executors.

The topology is then supported by the Zookeeper cluster where the master node will distribute code among worker nodes for execution (Amakobe, 2016).

The Storm architecture, is shown in Figure 7. It consists of a primary node Nimbus, a number of slave supervisors, and a Zookeeper cluster.

The master node of cluster is Nimbus, responsible for executing the topology and monitoring the execution of all process and Zookeeper cluster. It analyzes the topology and the task to be performed. It will then distribute the task to an available supervisor (Point, 2019). It consists for distributing data among all the worker nodes, assign tasks and monitoring failures. Nimbus and supervisors communicate with each other through a Zookeeper cluster.

A Zookeeper cluster is used to coordinate the work between the master node and the slave nodes (Nasiri, Nahesi and Goudarzi, 2019). It is responsible for managing all message communication, with the help of message acknowledgments, processing status, among others (Shahverdi, Awad and Sakr, 2019).

The cluster is capable of storing job topology information, slave supervisor status, cluster-wide state and configuration information (Sun *et al.*, 2019).

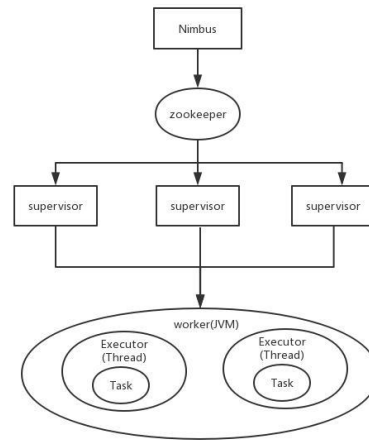


Figure 7: Storm Architecture (Sun *et al.*, 2019).

A worker creates executors and asks them to perform a particular task. Its process will have various executors. Each worker node runs a daemon called Supervisor, that can run one or more worker processes (Nasiri, Nahesi and Goudarzi, 2019).

2.5.2 Storm Advantages and Limitations

There are several advantages of Storm (Point, 2019):

- Storm is unbelievably fast because it has enormous power of processing the data;
- Storm has operational intelligence (it focuses on real-time dynamic, business analytics delivering visibility into data, streaming events and business operations);
- It can guarantee data processing if a process is killed by any of the connected nodes in the cluster or if messages are lost.

Storm has some drawbacks (Sun *et al.*, 2019):

- Resource allocation doesn't take the structural features of the task topology into account and cannot be adapted to the dynamic changes of the data load;
- The scalability of the system is limited.

3 COMPARISION OF BIG DATA STREAMING PLATFORMS

In this section, some features are analyzed to compare the presented platforms. In order to choose the main characteristics of big data streaming platforms, an

analysis of possible problems that companies had already had was made.

According to (Nasiri, Nahesi and Goudarzi, 2019), the characteristics many companies had difficulty are scalability, privacy, load balancing, fault tolerance, integration, consistency, timeliness, privacy, accuracy, among others. Then, it was concluded that these are the main characteristics in the analysis of big data streaming. Another analysis made on platform resources is that any platform needs to be robust, i.e., it contains the main characteristics of a big data streaming platform. It is necessary that it has a simple dashboard, is accessible anywhere.

The following features were selected for comparative analysis (Kolajo, Daramola and Adebisi, 2019) (Immanuel, 2019):

- *Fault-tolerance*: that allows an application to continue working without interruption;
- *Scalability*: that means research efforts should be focused on developing scalable structures that accommodate data flow computation mode, effective resource allocation strategy, and parallelization issues to address the increasing size and complexity of data;
- *Robustness*: it's the ability of a computer system to handle errors during execution;
- *Dashboards*: make it possible to visualize data in the form of graphs or images that show the most important graphics;
- *Integration*: it enables efficient operations on different data sets;
- *Consistency*: achieving high consistency (i.e. stability) in big data stream computing environments is non-trivial as it is difficult to determine which data is needed and which nodes should be consistent;
- *Security*: it proposes techniques for protecting a dataset before its analysis;
- *Time handling*: 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;
- *Stream SQL*: it's a query language that extends SQL and process real-time data streams;
- *ETL Optimization*: is the process by which data is extracted from optimized data sources;
- *Machine Learning*: data analysis method that automates the construction of analytical models;
- *Elasticity*: the degree to which a system is able to adapt to workload changes.

After choosing the attributes we will proceed to the comparison of the five big data streaming platforms, shown in Table 1.

Table 1: Platforms comparison based on the presented features.

Features	Flink	Kafka	Samza	Spark	Storm
Fault-tolerance	✓	✓	✓	✓	✓
Scalability	✓	✓	✓	✓	✓
Robustness	✓	✓	✓		✓
Dashboards	✓	✓	✓	✓	✓
Integration	✓	✓	✓	✓	✓
Consistency	✓	✓	✓	✓	
Security	✓	✓	✓	✓	✓
Time handling					✓
Stream SQL	✓	✓	✓	✓	✓
ETL optimization	✓	✓		✓	✓
Machine Learning	✓	✓	✓	✓	✓
Elasticity	✓	✓	✓	✓	✓

Flink, Kafka, Samza, Spark and Storm are open-source big data streaming platforms and are used for real-time data analysis. All of them offer fault-tolerance, scalability, dashboards, integration, security, SQL stream, machine learning and elasticity and have a simple implementation methodology.

Regarding the robustness, not all platforms offer this feature that is relatively important. We can verify that the only platform that doesn't contain this feature is Spark. Another feature that managed to divide the quality of the platforms was the consistency, four of the five platforms have this component, namely: Flink, Kafka Samza and Spark.

Finally, there is a slight highlight on the Storm, as only this tool contains time handling.

In general, there is an emphasis on three platforms, Flink, Kafka and Storm, because among the features chosen for comparison, these platforms form the ones that obtained the greatest number of features. These platforms only fail in one feature, as already mentioned. However, the Samza and Spark fail in two features, being just behind the other platforms.

4 CONCLUSIONS

As the amount of data generated by different devices worldwide is growing, flow processing becomes a crucial and essential requirement on big data streaming platforms. The main objective of this work was to describe and compare the most popular and open-source big data streaming platforms: Flink, Kafka, Samza, Spark and Storm. A description was made of these platforms, their architectures and advantages and limitations. The comparison was

made using features that were chosen through brainstorming and researches, taking into account the needs that companies have when using these big data streaming platforms. The Flink, Kafka, and Storm platforms were the ones that achieved the best range, as they contain more features that we analyzed.

As future work, we intend to choose three of the compared platforms to evaluate them with a benchmark application. Research on existing benchmarks will be carried out and the one that best fits to evaluate the platforms will be chosen. The evaluation will be made taking into account the features that have been compared. We intend to choose the best platform and use it in a real environment. An extensive quantitative assessment (performance) of these systems will also be a good suggestion.

REFERENCES

- Amakobe, M. (2016) 'A comparison between Apache Samza and Storm', *Colorado Tech University*.
- Behera, R. K., Das, S., Jena, M., Rath, S. K. & Sahoo, B. (2017). 'A Comparative Study of Distributed Tools for Analyzing Streaming Data', *2017 Int. Conference on Information Technology (ICIT)*, pp. 79–84.
- D'Silva, G. M., Khan, A., Gaurav & Bari, S. (2018) 'Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework', *2017 2nd IEEE Int. Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pp. 1804–1809.
- Foundation, A. S. (2019 a) Apache Kafka. Available at: <https://kafka.apache.org/>.
- Foundation, A. S. (2019 b) Apache Storm. Available at: <https://storm.apache.org/>.
- Freiknecht, J., Papp, S., Freiknecht, J. & Papp, S. (2018) 'Apache Kafka', *Encyclopedia of Big Data Technologies*. Springer, Cham, p. 8.
- Ghasemi, E. & Chow, P. (2019) 'Accelerating Apache Spark with FPGAs', 2016, *Wiley Online Library, Concurrency and Computation: Practice and Experience*, v31, Issue 2.
- Gurusamy, V., Kannan, S. and Nandhini, K. (2017) 'The Real Time Big Data Processing Framework Advantages and Limitations', *Int. Journal of Computer Sciences and Eng.*, 5(12): pp 305-312.
- Hoseiny Farahabady, M. R., Dehghani Samani, H. R., Wang, Y., Zomaya, A. Y. & Tari, Z. (2016) 'A QoS-aware controller for Apache Storm', *2016 IEEE 15th Int. Symposium on Network Computing and Applications (NCA)*, pp. 334–342.
- Imanuel (2019) Top 20, free open source and premium stream analytics platforms. Available at: <https://www.predictiveanalyticstoday.com/top-open-source-commercial-stream-analytics-platforms>.
- Instaclustr (2019) Apache Kafka. Available at: <https://www.instaclustr.com/apache-kafka/#apache-kafka-advantages>.
- Katsifodimos, A. and Schelter, S. (2016) 'Apache Flink: Stream Analytics at Scale', *2016 IEEE Int. Conference on Cloud Eng. Workshop (IC2EW)*, pp. 193–193.
- Kirillov, A. (2016) Apache Spark. Available at: <http://datastrophic.io/tag/spark/>.
- Kleppmann, M. (2018) 'Apache Samza', *Encyclopedia of Big Data Technologies*. SpringerLink, p. 8.
- Kleppmann, M. and Kreps, J. (2015) 'Kafka, Samza and the Unix Philosophy of Distributed Data', *IEEE Data Engineering Bulletin, December 2015*, 38(4), pp.4–14.
- Kolajo, T., Daramola, O. and Adebisi, A. (2019) 'Big data stream analysis: a systematic literature review', *Journal of Big Data volume 6, Article number: 47 (2019)*.
- Levy, E. (2019) 7 Popular Stream Processing Frameworks Compared. Available at: <https://www.upsolver.com/blog/popular-stream-processing-frameworks-compared>.
- Nasiri, H., Nahesi, S. and Goudarzi, M. (2019) 'Evaluation of Distributed Stream Processing Frameworks for IoT Applications in Smart Cities', *Journal of Big Data volume 6, Article number: 52 (2019)*.
- Neves, P., Bernardino, J. (2015) 'Big Data Issues', In *Proceedings of the 19th International Database Engineering & Applications Symposium (IDEAS '15)*, ACM, New York, USA, pp. 200–201.
- Point, T. (2019) Apache Storm. Available at: https://www.tutorialspoint.com/apache_storm.
- Safaci, A. A. (2017) 'Real-time processing of streaming big data', *Real-Time Systems*, v. 53, pp. 1–44.
- Shaheen, J. A. (2017) 'Apache Kafka: Real time implementation with Kafka architecture review', *Int. Journal of Advanced Science and Technology*, pp.35-42.
- Shahverdi, E. (2018) 'Comparative Evaluation for the Performance of Big Stream Processing Systems', *Int. Journal of Pure and Applied Mathematics, V. 119 No. 16*, pp.937-948.
- Shahverdi, E., Awad, A. and Sakr, S. (2019) 'Big Stream Processing Systems: An Experimental Evaluation', *2019 IEEE 35th Int. Conference on Data Eng. Workshops (ICDEW)*, pp.53-60.
- Shoro, A. G. and Soomro, T. R. (2015) 'Big Data Analysis: Apache Spark Perspective', *Int. Journal of Technical Innovation in Modern Engineering & Science (IJTIMES)*, V.4, Issue 5.
- Stratosphere, A. F. and Markl, B. V. (2018) 'Mosaics in big data', *DEBS '18: The 12th ACM Int. Conference on Distributed and Event-based Systems*, pp. 7–13.
- Sun, G., Song, Y., Gong, Z., Zhou, X. & Bi, Y. (2019) 'Survey on streaming data computing system', *ACM TURC 2019: ACM Turing Celebration Conf.*, pp. 1–8.
- Team, D. (2019) Apache Kafka Tutorial. Available at: <https://data-flair.training/blogs/apache-kafka-tutorial/>.
- Vaidya, N. (2019) Apache Spark Architecture – Spark Cluster Architecture Explained. Available at: <https://www.edureka.co/blog/spark-architecture/>.