International Journal of Scientific Research and Engineering Development-– Volume 4 Issue 3, May -June 2021

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

# A Survey of Data Stream Processing Frameworks

Sai Swarna Dowley Rajendran*, Pavithra H**

*(Department of Computer Science and Engineering, Rashtriya Vidyalaya College of Engineering, Bengaluru
Email: dowleyswarna@gmail.com)

** (Department of Computer Science and Engineering, Rashtriya Vidyalaya College of Engineering, Bengaluru
Email: pavithrah@rvce.edu.in)

----------**********----------

## Abstract:

Stream Processing is a technology widely used in fetching continuous streams of data within a small timeframe. In the internet era, data is not restricted to a small set of producers and consumers. There are a huge number of producers and consumers. One of the challenges in cultivating a streaming investigation foundation is the difficulty in picking the right stream handling system for the different use cases. Apache Kafka is a fault-tolerant, scalable and open-source distributed event streaming platform utilized by many organizations for streaming analytics, data integration and many crucial applications. Apache Flink is an open-source stream processing platform that is scalable, distributed, fault-tolerant, and stateful. Apache Storm is a fault-tolerant, distributed stream processing computational system for processing data streams. Apache Spark is an open-source unified analytics engine that supports batch, streaming, and iterative data processing. All these stream processing frameworks have their own advantages and disadvantages. This paper performs an in-depth comparative of these stream processing frameworks, which are broadly used to deal with huge measures of data. Such a study will be helpful in identifying the suitable stream processing framework for the required application.

*Keywords* **—stream processing, data streams, Kafka, Flink, Storm, Spark.**

----------**********----------

## I. INTRODUCTION

The ability to measure and deal with continuous streams of data is becoming a crucial part in developing a data-driven organization. Data streams are successions of unbounded tuples created consistently on schedule. Handling enormous volumes of data in batches is regularly not adequate in situations where new data must be prepared quick to rapidly adjust and respond to changes. Hence, data stream processing has acquired critical consideration. Software industry enterprises frequently propose a necessity to make data communication to be fast, highly reliable and fault tolerant. Enterprises have turned to technologies that respond to data at the time at which it is created for a variety of use cases and applications.

Historically, information was typically handled in groups dependent on a schedule or some specified threshold. However, the speed of information has sped up and volumes have swelled, and there are many use cases for which batch processing simply doesn't cut it. Stream processing allows applications to respond to new data records as and when they are received. Rather than grouping data and collecting it at some predetermined interval like batch processing, stream processing applications collect and process data immediately as they are generated. Stream processing are most often used in applications where data is generated continuously as a series of records, such as data from IoT sensors, application logs, server logs and transaction systems. Common paradigms include source/sink

  

and publisher/subscriber. Data records or events are generated from the source or publisher and are sent to streaming applications where the data is processed and then sent to sink or subscriber.

Presently numerous softwaretoolsexist for information ingestion, preparing, ordering, putting away and overseeing of information streams which makes it hard to pick the right blend of systems and devices for building streaming analytics

.

## II. LITERATURE SURVEY

Distributed systems for processing and analyzing Big data and event streams have piqued the interest of researchers since they are becoming a powerful instrument for various applications. [1] Jeyhun Karimov et al. examined the capabilities and performance of Apache Storm, Apache Spark, and Apache Flink, Stream Data Processing Systems (SDPs). [2] Tyler Akidau et al studied MillWheel which is a low-latency data-processing application that is often compared with Apache open-source systems for Stream Data Processing. [3] Aleksandra I. Stojnev and Dragan H. Stojanovi provide an overview of present systems and technologies in the field of data streams processing and analysis comparing spark streaming, storm, and flink against various aspects, including stream abstraction, Data flow abstraction which are main abstractions used to represent streams, native Machine Learning library along with the language used for the development of the framework, as well as the availability of different cloud platforms in support for application development. From the perspectives of use cases, architectures, fault tolerance, and licensing, [4] Xiufeng Liu et al. researched on of Stream Data processing systems, including Hadoop Online, Impala, S4, HStreaming, Storm, Flume, Spark Streaming, Kafka, Scribe, S4 and the relevant messaging technologies.

Unlike the other articles, this one discusses the architectural variations and distinct abstractions in Apache Kafka, Flink, Storm, and Spark Streaming when it comes to stream processing and analysis. Furthermore, these frameworks are compared in terms of application development and functionality.

## III. OPEN-SOURCE STREAM PROCESSING FRAMEWORKS

This section goes through some of the most popular open-source stream processing tools for industrial purposes, including Apache Kafka, Apache Flink, Apache Storm and Apache Spark

### *A. Apache Kafka*

The Apache Kafka is an open-source distributed event streaming platform that is developedin Scala and Java.

Kafka is a distributed messaging system that works in a cluster on numerous machines. These cluster of servers, each of which is called broker. Brokers keep communication streams organized into Topics. For load balancing, a topic is separated into partitions. Brokers are responsible for distributing partitions. A message sequence is contained in each partition. Each message is given an offset, which is a sequential id number that is unique inside its partition. The offset is used to identify each message within a partition. Messages are added by clients.

The Kafka cluster categorizes and stores a stream of records into topics. A timestamp, a key, and a value make up each record. Producer API, Consumer API, Connector API, and Streams API are the four key APIs supplied by Kafka.

A service can use the Producer API to broadcast a stream of messages to Kafka topic(s). A service may use the Consumer API to subscribe to a collection of topics and process the messages that are sent to them. The Connector API allows you to run reusable producers and consumers to link Kafka topics to other applications or data systems. The Streams API transforms the service into a stream processor, which consumes an input stream from topic(s) and outputs a stream to an output topic (s).

Any read/write operation is routed through the leader.The leader is in charge of keeping the other clones up to date. If one of the leaders fails, a

  

replica takes control.Apache Kafka can be used as a storage system, a messaging system, or a stream processing system.

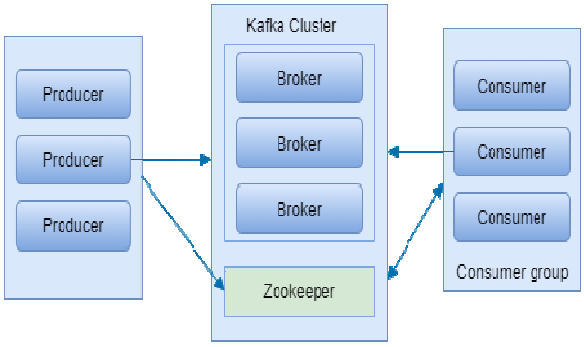


Fig. 1 Apache Kafka architecture

### B. *Apache Flink*

Apache Flink is an open-source, unified stream-processing and batch-processing framework that is developed in Java and Scala.

Machine learning, query processing, graph processing, batch processing, and stream processing are the tasks that the Flink engine can handle. Flink is made up of intermediary layers and an optimizer that makes it easier to construct complicated systems with improved performance. It can communicate with other systems such as HDFS and Apache Kafka. As a result, it makes it easier to interpret data from a variety of sources. It also contains a resource manager that controls the cluster's resources and gathers information on the jobs that have been completed and those that are still running. To fulfil its design aims, Flink uses numerous layers, which will be detailed later. It also uses a fault-tolerant algorithm that is lightweight and helps to improve the overall system performance.

Flink ensures that each record is only processed once. As a result, it entails buffering more data in a long-term storage system like Apache Kafka.

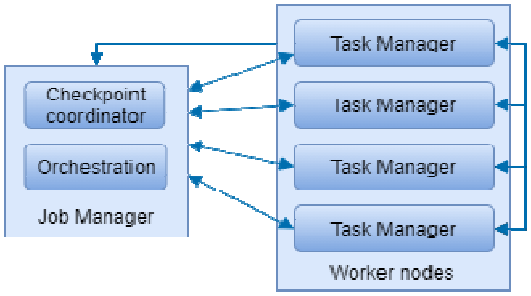


Fig. 2 Apache Flink architecture

### C. *Apache Storm*

Apache Storm is a free, fault-tolerant and open source distributed stream processing computation framework, written in Java and Clojure programming language.

Storm provides a real-time analytics, deep learning, and unbounded stream processing computing framework. It will accept messages that are continually generated and send them to multiple systems. Storm is built on a master-slave system. Nimbus is a master server that runs on a single node known as master node. On each worker node, there is a slave service named supervisor working. Supervisors start one or more worker processes, known as workers, that process the input in parallel. The output of worker operations is saved to a database or file system. Storm uses Zookeeper to coordinate distributed processes.

Storm has two kinds of input stream processing components: spouts and bolts. Spouts generate streams of tuples by processing external data. Spouts generate tuples, which are then sent to bolts. Bolts take tuples from input streams and generate tuples as inputs. Bolt input streams may come from spouts or another bolt. Spout is a Storm component that ingests data and generates a stream of tuples for the bolts to process. Storm Bolt takes tuples from spouts and processes them before sending them to external structures or other bolts.

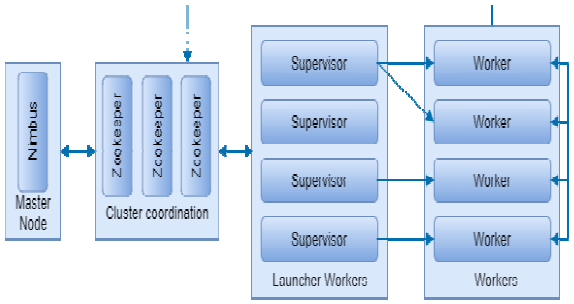


Fig. 3 Apache Storm architecture

### *D. Apache Spark*

Apache Spark is a large-scale data processing open-source unified analytics engine written in Scala. It provides high-level APIs in Java, Scala, Python and R programming languages, as well as an optimized engine that supports general execution graphs. It also includes a large number of higher-level tools, such as Spark SQL for SQL and structured data analysis.

At the most basic level, an Apache Spark programme is made up of two parts: a driver that transforms user code into several tasks that can be spread across worker nodes, and executors that operate on those nodes and carry out the tasks allocated to them.

Apache spark can support batch, iterative and streaming data processing. Spark will operate in a standalone cluster mode, requiring only the Apache Spark module and a JVM on each computer in the cluster. It can also runon Kubernetes, Docker swarm and Apache Mesos.The architecture of Spark Streaming framework is given in Fig. 4.

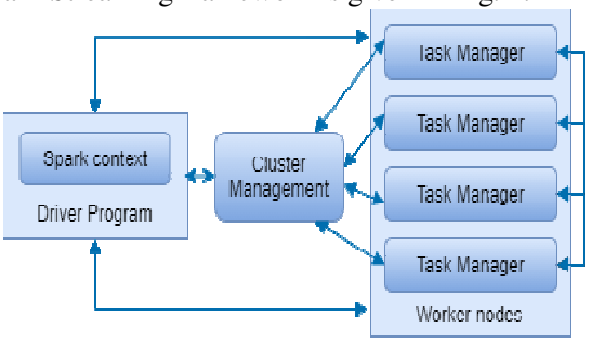


Fig. 4 Apache Spark Streaming architecture

## IV. COMPARISON

An outline of the inspected frameworks usefulness for stream handling, investigation, and application improvement is appeared in Table I.

TABLE I

| Criteria | Kafka | Flink | Storm | Spark Streaming |
|---|---|---|---|---|
| Language | Scala, Java | Scala, Java | Clojure, Java | Scala, Java, Python, R |
| Delivery semantic | Exactly once | Exactly once | Atleast once | Exactly once |
| Stream Abstraction | Kafka Streams | DataStream | Tuple | DStream |
| Execution model | Continuous streaming | SQL, batch micro-batch | Micro-batch | Micro-batch |
| Fault tolerance | Stream replay | Checkpoint | Checkpoint, stream replay | Checkpoint |
| Latency | Low | Low | Low/Medium | Medium |
| Throughput | High | High | Low | High |
| Resource Management | Standalone, YARN | YARN, Mesos, Standalone | YARN, Standalone | YARN, Mesos, Standalone |
| Data flow | DAG | CDG | DAG | DAG |
| State-fullness | Yes | Yes | No | Yes |
| API | Declarative | Compositional | Compositional | Declarative |
| Maturity | Medium | Medium | High | High |

## V. CONCLUSION

This paper presents a general study and comparison of the four major open-source distributed stream processing frameworks: Apache Kafka, Apache Flink, Apache Storm, Apache Spark Streaming, concentrating on their main concepts and architectural differences. The key abstractions used to represent streams, as well as the language used to construct the framework, along with the

various failure mechanisms and message semantics, are analyzed.

Each of the frameworks examined has its own set of benefits and drawbacks. The enormous variety of systems accessible is beneficial, but it presents a significant difficulty in terms of picking the appropriate components or processing framework for various use cases. Understanding the necessary capabilities of streaming architectures is critical for selecting the best design or usage option. Different abstractions are just one consideration among many, including performance, security, and tool and library interoperability. As these systems grow and emerge, we will expand our study to encompass diverse parts of them, as well as new frameworks.

## REFERENCES

[1] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen and V. Markl, "Benchmarking Distributed Stream Data Processing Systems," 2018 IEEE 34th International Conference on Data Engineering(ICDE),2018,pp.1507-1518,doi: 0.1109/ICDE.2018.00169.

[2] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. 2013. MillWheel: fault-tolerant stream processing at internet scale. Proc. VLDB Endow. 6, 11 (August 2013), 1033–1044. DOI:https://doi.org/10.14778/2536222.2536229.

[3] A. I. Stojnev and D. H. Stojanović, "Software systems for processing and analysis of big data and event streams," 2017 13th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS), 2017, pp. 128-131, doi: 10.1109/TELSKS.2017.8246245.

[4] Xiufeng Liu, Nadeem Iftikhar, and Xike Xie. 2014. Survey of real-time processing systems for big data. In Proceedings of the 18th International Database Engineering & Applications Symposium (IDEAS '14). Association for Computing Machinery, New York, NY, USA, 356–361. DOI:https://doi.org/10.1145/2628194.2628251.

[5] X. Liu, N. Iftikhar, X. Xie, "Survey of Real-time Processing Systems for Big Data", In Proceedings of the 18th International Database Engineering & Applications Symposium, July 2014, pp. 356-361.

[6] P. Carbone Asterios Katsifodimos, S. Ewen Volker Markl, S. Haridi Kostas Tzoumas, "Apache FlinkTM: Stream and Batch Processing in a Single Engine," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 36, no. 4, 2015

[7] M. Gorawski, A. Gorawska and K. Pasterak, "A survey of data stream processing tools" in Information Sciences and Systems, Cham, Switzerland:Springer, pp. 295-303, 2014.

[8] X. Liu, N. Iftikhar and X. Xie, "Survey of real-time processing systems for big data", *Proc. 18th Int. Database Eng. Appl. Symp*, 2014

[9] M. P. Singh, M. A. Hoque and S. Tarkoma, "A survey of systems for massive stream analytics" in arXiv:1605.09021, May 2016.

[10] G. Hesse and M. Lorenz, "Conceptual survey on data stream processing systems", *Proc. IEEE 21st Int. Conf. Parallel Distrib. Syst. (ICPADS)*, pp. 797-802, Dec. 2015.

[11] B. Yadranjiaghdam, N. Pool and N. Tabrizi, "A survey on real-time big data analytics: Applications and tools", *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, pp. 404-409, Dec. 2016.

[12] T. Kolajo, O. Daramola and A. Adebiyi, "Big data stream analysis: A systematic literature review", *J. Big Data*, vol. 6, no. 1, 2019.

[13] M. Gehring, M. Charfuelan and V. Markl, "A comparison of distributed stream processing systems for time series analysis", *Proc. BTW Workshopband*, pp. 205-214, 2019.

  