Indian Journal of Science and Technology, Vol 9(14), DOI: 10.17485/ijst/2016/v9i14/84797, April 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Efficiency of Stream Processing Engines for Processing BIGDATA Streams

B. V. S. Srikanth and V. Krishna Reddy

Department of Computer Science and Engineering,
K L University, Green Fields, Vaddeswaram,
Guntur - 522502, Andhra Pradesh, India.
srikanth.bhuvanagiri@kluniveristy.in,
vkrishnareddy@kluniversity.in

Abstract

Background/Objectives: In this paper, there are real-time stream & big data stream applications running on a large amount of clusters & live migration of data using intensive & interactive tasks. **Methods/Statistical Analysis:** Traditionally, there are many stream processing engines which are going to manage the processing data in this streaming scenario, ongoing stream processing the data must be handle due to faults & stragglers within a second-scale latency. At present, there are efficient stream processing engines which can avoid these faults & stragglers by using the stream processing engines like APACHE SPARK & APACHE FLINK. **Findings:** In the real-time data stream processing there are some problem occurrences while processing the data, those are data latency, fault-tolerance, mutable data-sets and more stragglers. To avoid all these issues there is an efficient stream processing engines with additional added mechanisms like dolly retreat mechanism for avoiding stragglers and process data efficiently. **Application/Improvements:** Now a day's demand for stream processing is increasing a lot which may be real-time streaming or normal cluster data transformation. Data has to be processed fast, so that companies are integrating these stream processing engines to changing their business conditions effectively in real-time for analyze their data more effectively with help of Fast Stream Processing Engines.

Keywords: Fault-Tolerance, RDD, Second-Scale Latency and Immutable Datasets, Stragglers

1. Introduction

Stream is a process of bulk portion of data. For streaming traditionally we follow store-then-process paradigm, later we come up with Stream Processing[1]. Stream processing is a transition of bulk amount of data, using continuous Ad-hoc queries. Stream processing is designed to analyze and stand to process the bulk or small amount of data based upon the continuous data queries, intensive & interactive tasking queries and real-time processing in Big Data applications. Stream processing is mainly designed to handle the data transmission between the Clusters, Servers, IDE, Connectors, and Analytics without any occurrences of fault-tolerance[2,3] & Stragglers[4]. Stream processing is developed to handle the faults & stragglers within the second-scale latency, high volume of scalability in real-time and high availability of data without any congestion occurrences.

At Present in the real-time scenario for the Big Data Stream processing had to face so many challenges:

- Processing the bulk massive amounts of stream events.
- Real-time streaming responsiveness to handle the various changing market conditions.
- Performance of Stream processed data & maintain High availability at real-time.
- Integration issues of several plug-ins to handles the small datasets to process the data stream.
- At large volume datasets maintain the scalability for stream processing data due to Ad-hoc queries.
- Live analytics for streaming the conference media at real-time data discovery, monitoring and continuous query processing with automated alerts & reactions.

*Author for correspondence

In the Stream Processing[5] for the Big Data applications in the technical perspective we have to face the challenges and implements in the Servers, develop a programmatic approach for IDE, & should maintain respective.

Connectors & Analytics and finally manage, monitor, automated alert & reactions for Streaming Analytics and also maintain the Live Streaming for business operational intelligence which aggregates stream data for graphs, charts, slice and so on.

2. Prerequisties

In this paper there is a discussion about various Big data Stream processing Methodologies[6] for stream processing as follows:

2.1. Big Data Streaming:

Streaming is a process of programming API, for processing the bulk amount of datasets without any faults. For stream processing there are different types: AURORA[7], Millwheel[8] & BOREALIS[9]. These stream processing engines are distributed processing purpose.

2.2. Real-Time Streaming:

Real-time Stream processing is the transmitting the data at live based on the continuous user ad-hoc queries. It means real-time data which should be managed & processed automated due to alerts & reactions. In Real-time streaming keep move on data, handle stream imperfections, Partition & scalable automatically, Process & Respond Instantaneously.

2.3. Batch Processing

Batch means running the user executed query in a scheduled way. It means we process our datasets based on the Batch with some integration of Hadoop, storm, samza, spark & flink. In batch processing there will be a mechanism of map reduces style to process the data.

2.4. Stream Processing Engines (SPEs):

Stream Processing Engine is specifically designed for the stream-based application. These SPE is designed to process the streaming data based on the continuous queries. Stream Processing Engine consists of embedded DBMS for handling data without necessarily storing them & performs Stream-oriented processing logic.

2.5. In-Memory Data Processing:

The concept of in-memory data processing provides the high performance of complex event processing support & also supports the highly optimized event-window processing. These two processing support with delivery of low latency in order to strictly process the lossless data without necessarily storing it.

3. Problem Statement

Traditionally we processed the data based on store-then-process paradigm. Later, we define a methodology for streaming as Stream Processing Engine (SPEs). In many big data applications transforming a data between clusters, servers, nodes and for live data means real-time data processing for both of these stream processing we need to handle & overcome the problems of Fault-Tolerance[10], Faults & Stragglers within the second-scale latency, and maintain high-availability, data processed scalability with better performance of stream processing. For handling these problem there will be an approach & methodology for proposing a new methodology it will be explained in later section.

4. Proposing an Approach & Methodology

In this paper, the proposed model for the new approach & methodology for the stream processing of large amount of data between various servers, clusters, and data nodes & real-time processing. Therefore, the discussion on various traditional streaming techniques which are used for processing large amounts of big data using batch processing, stream processing & unified data intensive and interactive jobs has been takes place where explained as following criteria:

4.1. Map Reduce:

MapReduce[11] is a Framework, which consists of basically two phases Map & Reduce with process the data using batch processing style with applying sort & shuffle functionalities of data[12].

Map-Reduce consists of two separate and distinct phase.

- Mapper: A function from an input pair of key/value to a list of intermediate pairs of key/value

Map: (key-input, value-input) List (key-map, value-map)

- Reduce: A function from an intermediate pairs of key/value to a list of output pairs of key/value13.

Reduce: (key-map, list (value-map)) list (key-reduce, value-reduce) as shown in Figure 1.

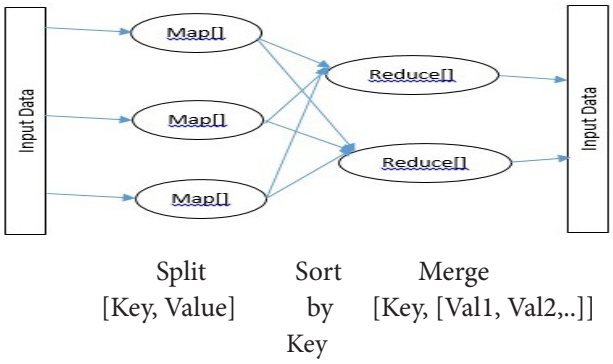


Figure 1. Flow of Map-Reduce Architecture.

Problem: In the Map-reduce framework the problem is when there is a data processing using batch processing, it cannot handle the multiple & short-live queries for requests and responses & also it cannot handle the Stragglers.

4.2. Apache Kafka:

Apache Kafka is a pub/sub system developed by LinkedIn. Which describes the partitioned for data and it provides replication & distribution based of log-based service. Kafka maintains the *topics* for *consumer* to feed messages and subscribing the topics will *publish* message feeds as *producer* due to the requests & responses of Kafka cluster[14] as shown in Figure 2.

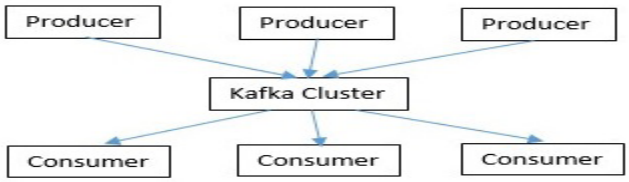


Figure 2. Apache Kafka Architecture.

Problem: In Kafka model, based on the Pub/Sub system it maintains cluster & handle faults but in high-level latencies it cannot handle Faults & stragglers issues.

4.3. Apache S4:

S4 is an approach for proposing a scalable, fault-tolerant, pluggable platform that allows programmers to easily develop applications for processing continuous unbounded streams of data which is distributed & general purpose. The S4 programing configuration is based on spring-xml configuration[15].

Problem: Problem with S4 Distributed platform is it cannot guarantees the message delivery subsystem. For huge amount of data it cannot handle Faults & Stragglers.

4.4. Apache Storm:

Apache Storm is for managing unbounded data streams over a distributed real-time streaming, developed by twitter. Mainly it varies from map-reduce is due to maintaining storm topologies. For example, a stream of tweets from twitter can be transformed into stream of trending topics. The two main components of Storm are tuples and nodes[16] as shown in Figure 3.

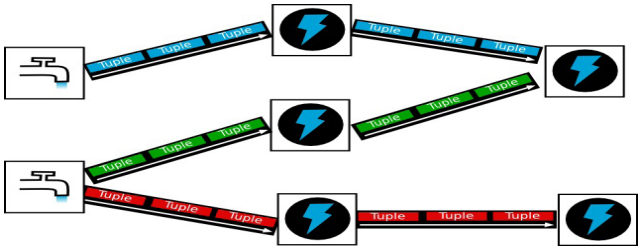


Figure 3. STORM Architecture.

Problem: In Storm even though it can process a million tuples per second per node. But, at node it hold the data to send upto the acknowledge had to be received it is a delay for data processing for stream data & it cannot handle Stragglers.

4.5. Twitter's Summing Bird:

Twitter Summing bird[17] is a framework for integrating batch processing for Hadoop/map-reduce & online map-reduce computation for Storm topologies for online processing of streams of data. The combination both Hadoop & Storm is called Hybrid model.

Problem: The problem with summing bird is it uses the re-computation f***or main***taining the past data based on the storm computation due to Hadoop and storm log-loading custom mechanism, it is harder to manage to log details & it cannot handle Stragglers.

4.6. Amazon Kinesis:

Amazon Kinesis[18] is for real-time data processing for distributed environment over large amount of data streams.

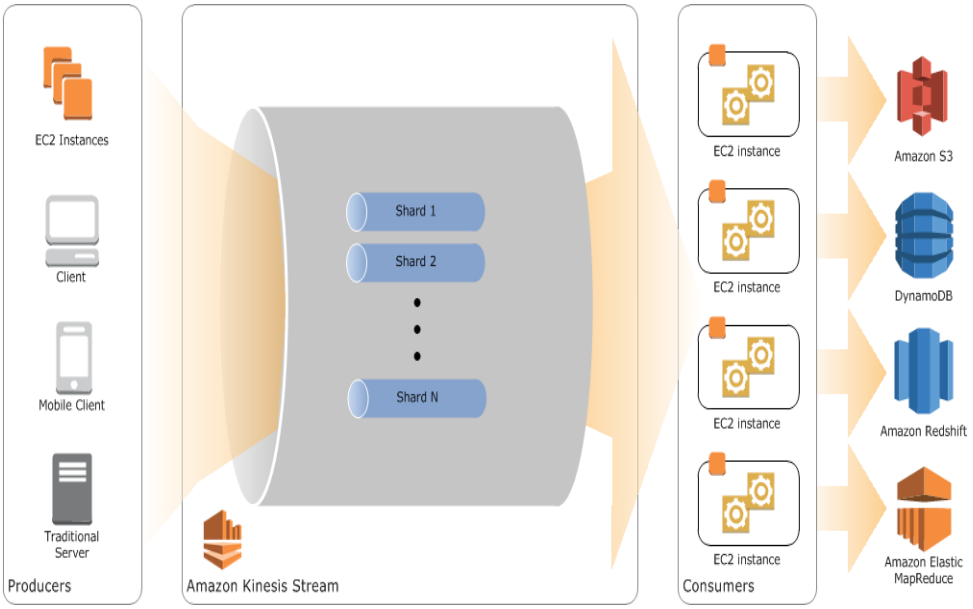


Figure 4. Amazon Kinesis Architecture.

Amazon kinesis is cloud-based service provide by Amazon Web Services (AWS) to store & process the TBs of data streams per hour continuously for Amazon Client Library (KCL) to process & manage the stream data based on the sources of web clicks, client required finance transactions, feeds for social media & log-based tracking events as shown in Figure 4.

Problem: The problem with Amazon Kinesis is that in real-time processing there will be low latency issues& it cannot handle Stragglers.

4.7. Data Torrent:

Data Torrent RTS[19] is an open source unified stream and batch processing engine to stream real-time data streaming with high performing, fault-tolerant, scalable, Hadoop-native in-memory platform. It can process billons of events per second and recovering node from outages with no data loss.

***Problem*:** For large amount of data streaming it can't handle stragglers.

4.8. Apache SAMZA:

Apache Samza is a distributed stream processing framework. It uses Apache Kafka for messaging, and Apache to provide fault tolerance, processor isolation, security, and resource management.

Samza processes ***streams***. A category which is immutable *messages* of data streams are in similar types as shown in Figure 5.

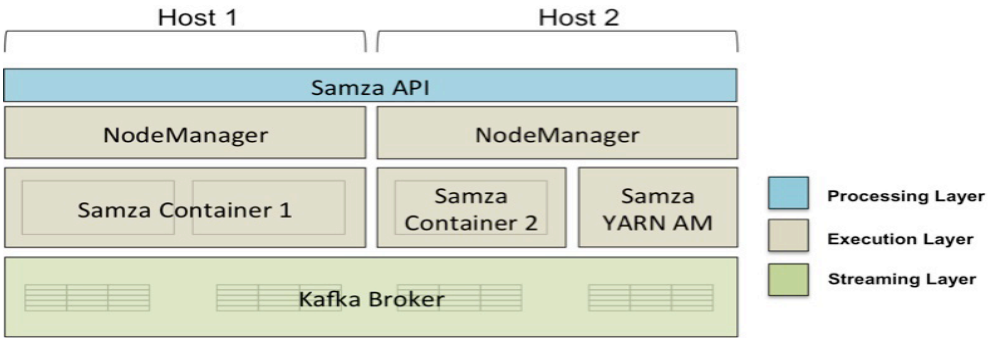


Figure 5. Architecture for SAMZA.

Samza is made up of three layers:

- A streaming layer.
- An execution layer.

Problem: Cannot handle stragglers.

4.9. Apache Spark:

Apache Spark™[20] is a stream processing engine for fast & large-scale processing of stream data. *Spark introduce D-Streams (Discretized streams) for processing the stream data using RDD based on in-memory processing concept.* Here RDD means Resilient Distributed Datasets[21] which process the stateless data worksets for handling fault-tolerance. In spark these RDD manages the streaming computations based on batches of results due to some speculation & handles Straggler using RDD[22] as shown in Figure 6.

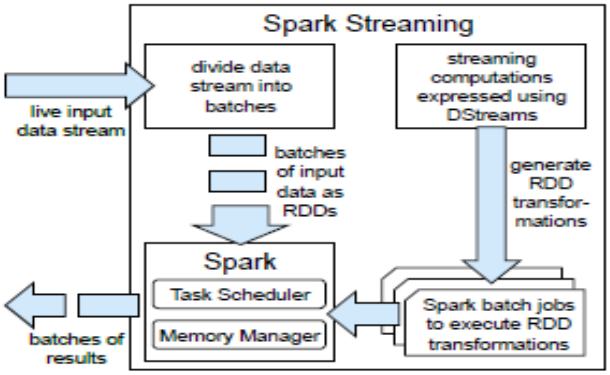


Figure 6. Architecture for High-level overview of Spark Streaming based on batches of input data as RDDs.

Spark Streaming providing the features for unified & batch processing as follows:

First, for the spark job we can combine D-streams with RDDs computations.

Second, using batch processing mode we run D-streams on past historical data. Means streaming report on past data.

Third, using Scala console D-streams run ad-hoc queries.
Problem: Spark is better than all stream processing engine and it can handle faults & stragglers, but using RDD in-memory concept it processing low streaming compare to FLINK streaming.

4.10. Apache Tez:

Apache Tez[23] is a Framework of batch and interactive data processing for data applications in coordination either Hadoop map-reduce & YARN support. Tez using map-reduce maintain scaling the petabytes of data.
By allowing projects like Apache Hive and Apache Pig to execute & perform DAG tasks to manage map-reduce job on tez support as shown in Figure 7.

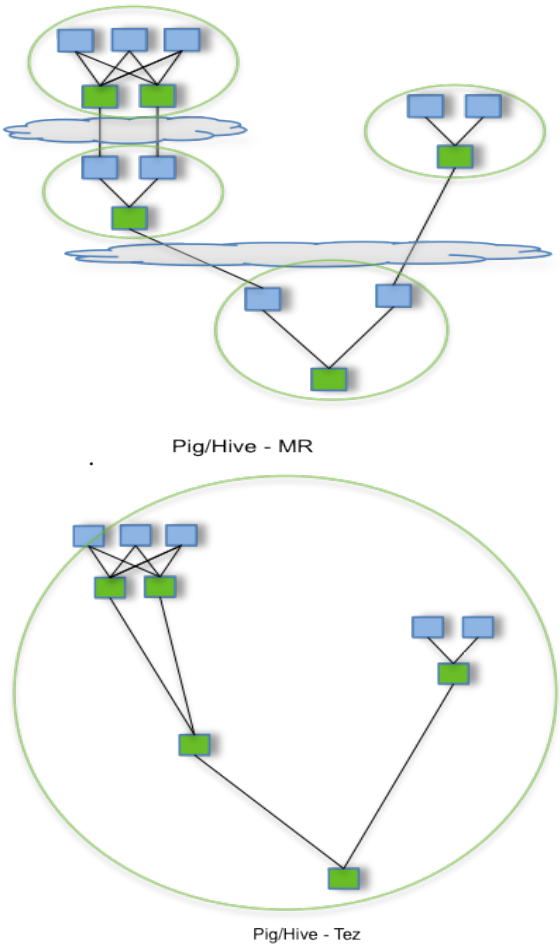


Figure 7. Architecture for Apache TEZ.

Problem: Cannot handle stragglers.

4.11. SQL Stream:

SQL Stream[24] is an innovative to solve real-time data stream processing. In Data stream processing the in-memory, record-by-record analysis of machine data in motion. For real-time data streaming it will take automated alerts & actions in order to maintain the faults & reacts as shown in Figure 8.

Problem: Can't handle stragglers.

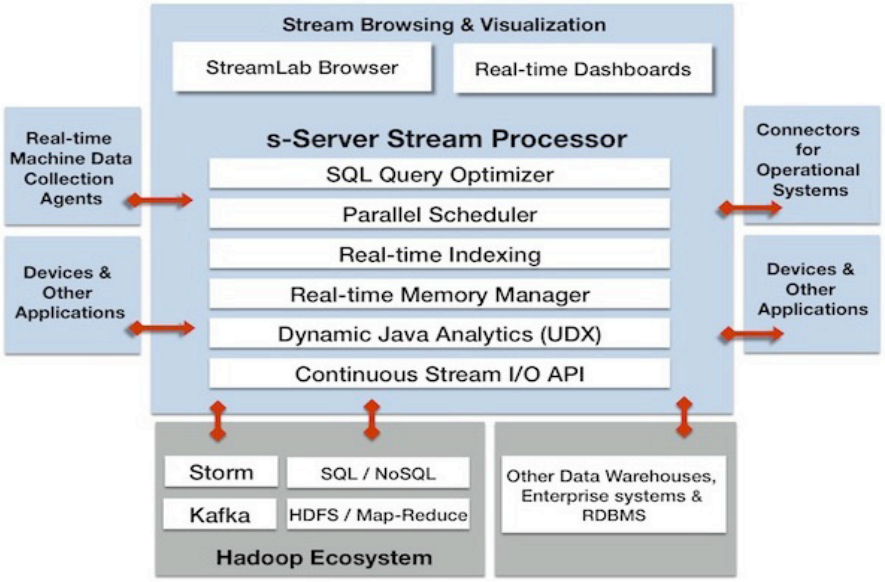


Figure 8. SQLStream Stream Processor Architecture.

4.12. Apache Flink

Apache Flink[25] is component of alternative approach for Hadoop map-reduce style for data processing system. It comes with its own runtime, rather than building on top of Map-Reduce. It is independent for the ecosystem of Hadoop. Therefore, in-built Hadoop cluster & YARN support is there for flink to processing the data in map-reduce style and also in iterative & intensive stream processing as shown in Figure 9.

Processes: When the Flink system is started, it bring up the *Job Manager* and one or more *Task Managers*. The Job Manager is the coordinator of the Flink system, while the Task Managers are the workers that execute parts of the parallel programs.

Apache Flink done two types of iterations based on Iterative Function. They are *Bulk* & *Delta* iterations as shown in Figure 10.

Flink Streaming has Iterate & Delta iterate approach, based on the iterative & intensive processing. Let us compare some iterate analysis in the following Table 1.

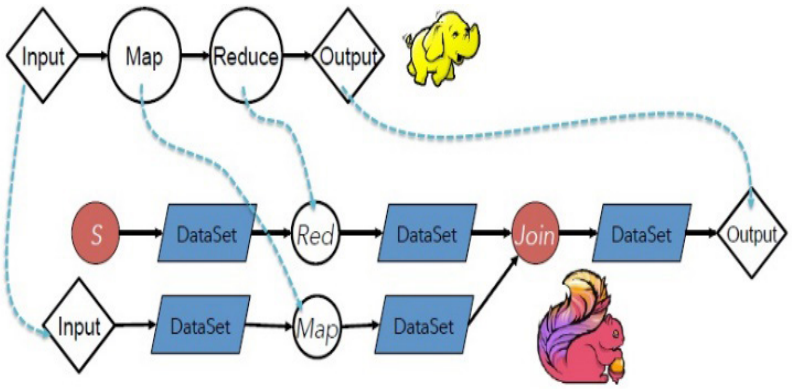


Figure 9. Flink Architecture.

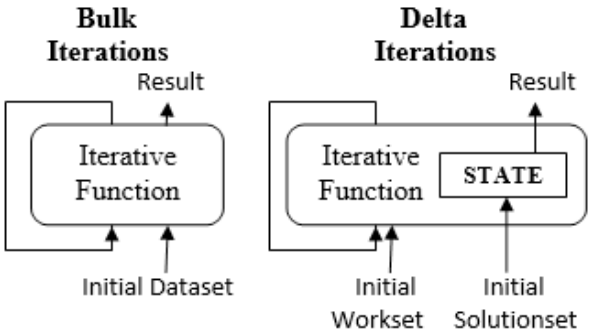


Figure 10. Two Types of Apache Flink Iterations.

Table 1. *Table for comparison of iteration in Flink model*

Input	Iterate	Delta iterate
Iteration Input	Partial solution	Work set and solution set
Step Function	Arbitrary Data Flows	
State Update	Next partial solution	Next work set Changes to solution set
Iteration Result	Last partial solution	Solution set state after last iteration
Termination	Maximum number of iterations(default) Custom aggregator convergence	Maximum number of iterations or empty work set(default) Custom aggregator convergence

5. Comparisions

5.1. Comparison with SQLstream vs. Spark

SQLstream is the most scalable stream processor available and offers the highest ingestion rate into Hadoop of any stream processing platform. In this comparison for Hadoop ingestion & streaming application for Storm, Spark & SQLstream. In these analysis SQLstream will process more data rate based on the Hadoop ingestion (MB/s) than storm & flume. In streaming application the stream of data in events/sec/cores in which the SQLstream is more efficient once than spark & storm as shown in Figure 11.

SQLstream also offers the fastest and most efficient platform for ingesting data into Hadoop to support rates up to 448 MB/s & for streaming applications the processed once must be higher.

5.2. Comparision Between Spark vs. Flink

Flink is optimized for cyclic or iterative processes by using iterative transformations on collections. Flink can process the stream data in Batch processing & Resilient Distributed datasets at same state. Flink streaming can be processed the large & small amount of data based on unified, batched, iterative & intensive processing task. Flink can process the data fastly and handling stragglers by the Dolly mechanism.

Spark on the other hand is based on resilient distributed datasets (RDDs). Spark streaming can process the

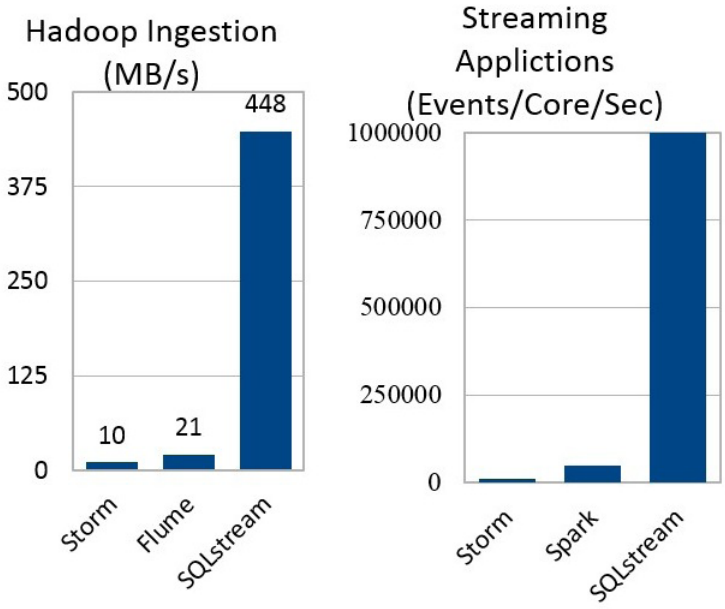


Figure 11. SQLstream Performance Benchmarks.

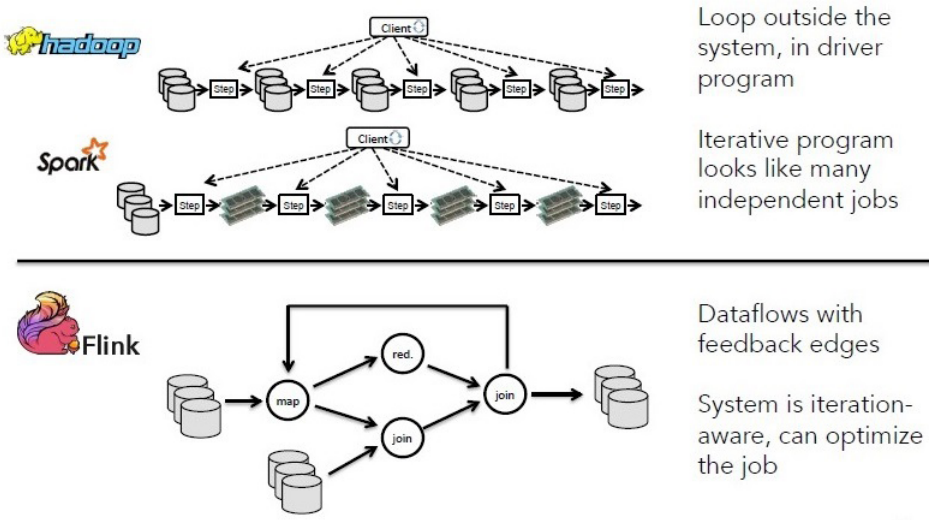
data without any stragglers based on in-memory concept handle by the Resilient Distributed Dataset (RDD).

Let see the built-in & driver-based iterations compared by Hadoop, spark with Flink as shown in Figure 12.

Based on the word count example comparing the spark the flink with its running time & number of iterations for graphical analysis report comparison as follows as shown in Figure 13:

For the Grep count example for count of application analysis here grep will be compared iteratively based on 5 terms the runtime will be calculated on seconds-latency & transmitted streamed log data will be in GBs. For which the graphical analysis is as follows as shown in Figure 14.

In this analysis to transmit 1TB of data Spark Streaming will take 37 minutes (as per sec it will be 2220s) of time to processed whereas, Flink streaming will process that whole 1TB data in 9minutes (as per sec it will be 540s). because in Spark the consists of RDD which 100: stored the data in the in-memory such that for streaming spark will starts spilling RDD to DISK. The graphical analysis will be shown as following graph as shown in Figure 15:



Built-In vs. Driver Based Iterations

Figure 12. Comparison iterations with Hadoop, Spark & Flink.

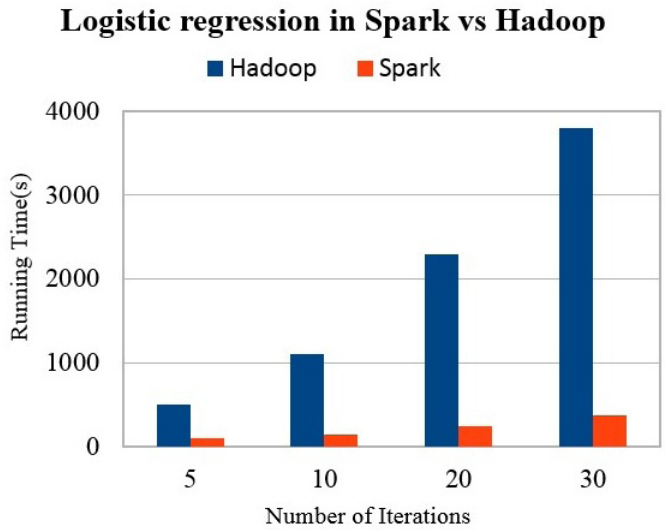


Figure 13. Comparing iteration-based analysis for Spark vs. Flink.

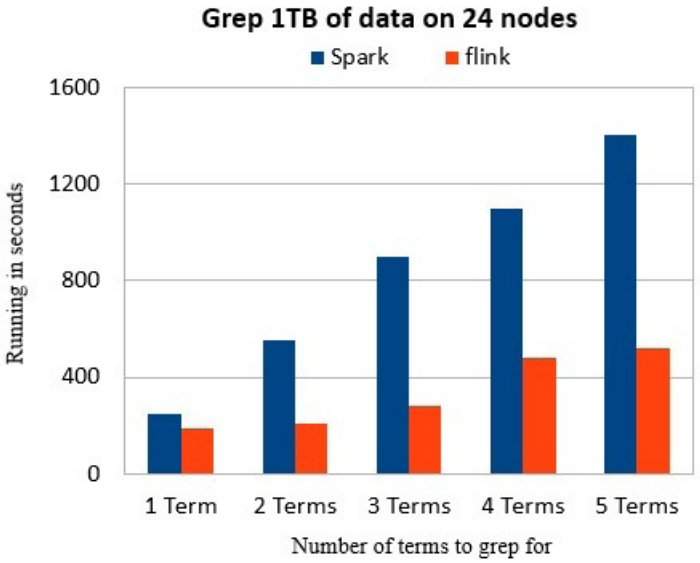


Figure 14. Grep count for Spark vs. Flink.

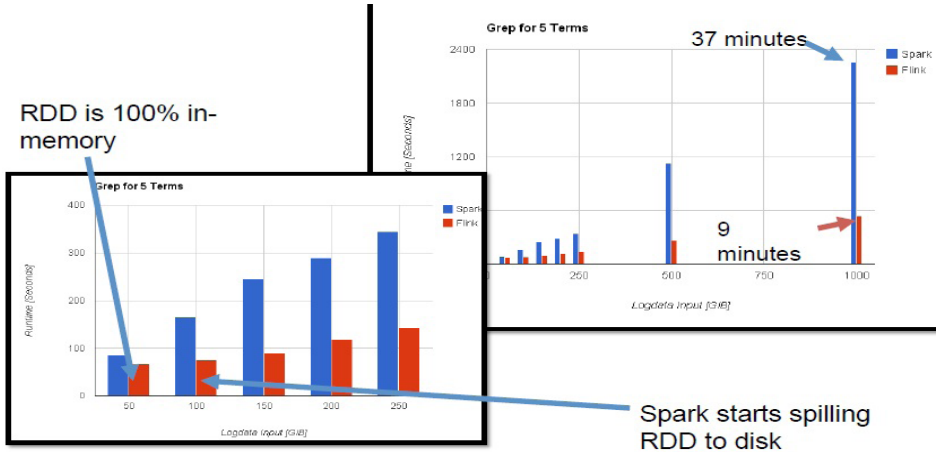


Figure 15. Grep count for Spark vs. Flink.

6. Conclusion

In this Paper, there is a discussion about various stream processing engines & the comparative study & their techniques represented in a pictorial representation about which is the effective & efficiency stream data processing engines. In this Stream Processing Engines the *Apache Spark & Apache Flink* are the efficient stream processing engine. They can handle the low-latency data, Fault-tolerance, maintains immutable datasets & finally they can avoid stragglers by adding addition mechanism *dolly retreat* using that they can avoid stragglers. The scope of this paper is to help the researches to find which is the effective & efficient stream data processing engine & also the future scope of this work is to using these efficient stream processing engine let us analyze the data using analytics tools with help of these stream processing and maintain the data without faults & stragglers.

7. References

1. Arasu A, Babcock B, Babu S, Datar M, Ito K, Nishizawa I, Rosenstein J and Widom J. STREAM: The Stanford stream data management system. SIGMOD. 2003.

2. Zaharia M, Das T, Li H, Shenker S and Stoica I. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. USENIX Association: In Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing. 2012; p. 10–10.
3. Martinaa M, Vaithiyanadhan V, Raghunadhan A. A Survey on Categorization of Fault Tolerance in Wireless Sensor Network. Indian Journal of Science and Technology. 2015 Dec; 8(35). Doi: 10.17485/ijst/2015/v8i35/86695.
4. Ananthanarayanan Ganesh, Ghodsi Ali, Shenker Scott, Stoica Ion. Effective Straggler Mitigation: Attack of the Clones. nsdi' 13, Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation. 2013; p. 185–98.
5. He B, Yang M, Guo Z, Chen R, Su B, Lin W and Zhou L. Comet: batched stream processing for data intensive distributed computing. In SoCC '10.
6. Babcock B, Babu S, Datar M, Motwani R and Widom J. Models and issues in data stream systems. ACM, Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2002; p. 1–16.
7. Abadi DJ, Carney D, Cetintemel U, Cher–niack M, Convey C, Lee S, Stonebraker M, Tatbul N and Zdonik S. Aurora: a new model and architecture for data stream management. The VLDB Journal. 2003; 12(2):120–39.
8. Akidau T, Balikov A, Bekiroglu K, Chernyak S, Haberman J, Lax R, McVeety S, Mills D, Nordstrom P and Whittle S. MillWheel: Faulttolerant stream processing at internet scale. In VLDB. 2013.
9. Abadi DJ, Ahmad Y, Balazinska M, Cherniack M, Hyon Hwang J, Lindner W, Maskey AS, Rasin E, Ryvkina E, Tatbul N, Xing Y and Zdonik S. The design of the borealis stream processing engine. In CIDR. 2005; p. 277–89.
10. Shanmugasundaram M, Kumar R, Mallikarjun Kittur Harish. Approaches for Transient Fault Tolerance in Multiprocessor—A State of Art. Indian Journal of Science and Technology. 2015 July; 8(15). Doi: 10.17485/ijst/2015/v8i15/55793.
11. Thirumala Rao B, Sridevi NV, Krishna Reddy V, Reddy LSS. Performance Issues of Heterogeneous Hadoop Clusters in Cloud Computing. 2011 May; 11(8) Version 1.0:81–87.
12. Dean J and Ghemawat S. Mapreduce: simplified data processing on large clusters. Commun. ACM. 2008 Jan; 51:107–13.
13. Ghemawat S, Gobioff H and Leung S-T. The Google File System. In Proceedings of SOSP'03. 2003.
14. Wang Guozhang, Koshy Joel, Subramanian Sriram, Zadeh Mammad, Narkhede Neha, Rao Jun, Kreps Jay, Stein Joe. Building a Replicated Logging System with Apache Kafka. In VLDB 2015, Proceedings of the VLDB Endowment. 2015 August; 8(12) p. 1654–55.
15. Neumeyer L, Robbins B, Nair A and Kesari A. S4: Distributed stream computing platform. In Intl. Workshop on Knowledge Discovery Using Cloud and Distributed Computing Platforms (KDCloud). 2010.
16. Kwon Y, Balazinska M and Greenberg A. Fault-tolerant stream processing using a distributed, replicated file system. Proceedings of the VLDB Endowment. 2008; 1(1):574–85.
17. Boykin Oscar, Ritchie Sam, O'Connell Ian and Lin Jimmy. Summingbird: A Framework for Integrating Batch and Online MapReduce Computations. In VLDB 2014. 2014 August; 7(13):1441–51.
18. Amazon Kinesis Streams Developer Guide. Data Accessed: 15/1/2016: Available from: http://docs.aws.amazon.com/kinesis/latest/dev/kinesis-dg.pdf.
19. DataTorrent RTS: Real-Time Streaming Analytics for Big Data. Data Accessed: 22/1/2016: Available from: https://www.datatorrent.com/resources/datatorrent-rts-real-time-streaming-analytics-for-big-data/.
20. Zaharia Matei, Chowdhury Mosharaf, Franklin Michael J, Shenker Scott, Stoica Ion. Spark: Cluster Computing with Working Sets. HotCloud'10, Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 2010; p. 10–10.
21. Zaharia Matei, Das Tathagata, Li Haoyuan, Hunter Timothy, Shenker Scott, Stoica Ion. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. SOSP'13, Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013; p. 423–38.
22. Zaharia M, Chowdhury T, Das A, Dave J, Ma M, McCauley M, Franklin S, Shenker and Stoica I. EECS Department, UC Berkeley: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Technical Report UCB/EECS-2011–82. 2011.
23. Sahah Bikas, Shahh Hitesh, Sethh Siddharth, Vijayaraghavanh Gopal, Murthyh Arun, Curinom Carlo. Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications. SIGMOD'15, Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. 2015; p. 1357–69.
24. SQL stream Blaze—A Streaming Data Hub for the Real-time Enterprise. Data Accessed: 29/1/2016: Available from: http://www.sqlstream.com/library/SQLstream_Blaze_ProductBrief.pdf.
25. Alexandro, Bergmann Rico, Ewen Stephan, Freytag Johann-Christoph, Hueske Fabian, Heise Arvid, Kao Odej, Leich Marcus, Leser Ulf, Markl Volker, Naumann Felix, Peters Mathias, Rheinlander Astrid, Sax Matthias J, Sebastian. The Stratosphere platform for big data analytics. VLDB'14. 2014; 23(6):939–64.