International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-7, Issue-6, March 2019

Distributed Programming Frameworks in Cloud Platforms

Anitha Patil

Abstract: ***Cloud computing technology has enabled storage and analysis of large volumes of data or big data. With cloud computing, a new discipline in computer science known as Data Science came into existence. Data Science is an interdisciplinary field which includes statistics, machine learning, predictive analytics and deep learning. It is meant for extracting hidden patterns from big data. Since big data consumes more storage space that cannot be accommodated with traditional storage devices, cloud computing resources of Infrastructure as a Service (IaaS) is used. Therefore, big data and big data analytics cannot exist without cloud computing. Another important fact is that big data can be subjected to analytics for obtaining Business Intelligence (BI). This process needs distributed programming frameworks like Hadoop, Apache Spark, Apache Flink, Apache Storm and Apache Samza. Without thorough understanding about these frameworks that run in cloud platforms, it is difficult to use them appropriately. Therefore, this paper throws light into a comparative study of these frameworks and evaluation of Apache Flink and Apache Spark with an empirical study. TeraSort benchmark is used for experiments.***

Keywords: Cloud computing, big data, big data analytics, distributed programming frameworks.

I. INTRODUCTION

Cloud computing technologies paved way for various other technologies grow over Internet. For instance with cloud computing resources big data and tools to handle big data are growing rapidly. There are many distributed programming frameworks available for handling big data. In other words, big data analytics is made with such frameworks [2]. Big data is the data with voluminous data which may increase from time to time rapidly and contain structured, unstructured and semi-structured data. There are different programming frameworks that support big data analytics with user-defined and pre-defined algorithms in distributed environment [17]. The distributed programming frameworks studied in this paper are Hadoop [16], [21], Apache Flink [1], [2], [3], [4], [5], Apache Spark [2], [3], [4], , [13], [14], [15], [16], Samza [6], [7], and Apache Storm [8], [10], [11], [16]. These frameworks are discussed and their architectural overview is presented in Section 2.

Revised Manuscript Received on 30 March 2019.
***** Correspondence Author
Anitha Patil, Department of Computer Science Engineering

Apache Airavatha [12] is another framework explored in [12]. Asynchronous message passing is another formwork used in distributed environments [19]. There are certain distributed dynamic data models as well [20]. Apart from this, two distributed programming frameworks are evaluated with comparative study. They are known as Apache Flink and Apache Spark. These frameworks are used for big data processing. TeraSort benchmark is used for empirical study. The observations are made with different performance metrics like execution time, network usage in terms of incoming and outgoing traffic and the throughput for read and write operations. Our contributions in this paper are as follows. Four distributed programming frameworks are provided with architectural overview. Then experiments are made with Apache Flink and Apache Spark in cloud environment. The TeraSort benchmark is used to evaluate the frameworks with different sizes of datasets. The remainder of the paper is structured as follows. Section 2 presents distributed computing frameworks. Section 3 presents experimental results. Section 4 concludes the paper and gives directions for future work.

II. DISTRIBUTED COMPUTING FRAMEWORKS

Distributed programming frameworks or distributed computing frameworks that play crucial role in big data processing are covered here. These frameworks provide pre-defined building blocks to developers to have faster application development. The framework will have execution flow pre-defined including task scheduling, fault tolerance and other aspects. The frameworks also provide programming models that can help in writing algorithms for data analytics. The following sub sections provide overview of different frameworks.

2.1 Hadoop

Apache Hadoop is an open source framework that implements MapReduce programming paradigm. This framework has associated file system known as Hadoop Distributed File System (HDFS). Apart from this it also has a scheduler known as YARN. The MapReduce paradigm is derived from functional programming. It has map and reduce functions. The former is assigned to multiple machines and latter is meant for aggregating the results of all map functions. WordCount is one of the benchmark applications to understand MapReduce. Hadoop stores data

in HDFS and delay scheduling is taken care of by YARN. YARN improves data locality and reduces latency.

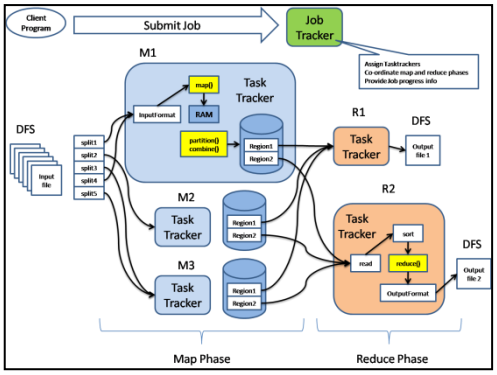


Figure 1: Architectural overview of Hadoop

As presented in Figure 1, it is evident that there are two phases known as Map phase and Reduce phase. Both inputs and outputs are associated with HDFS. HDFS follows a paradigm known as master and slave. Master takes care of authorization and data placement while slave maintains chunks of data and reports back to master from time to time. Slaves also maintain replicas for fault tolerance.

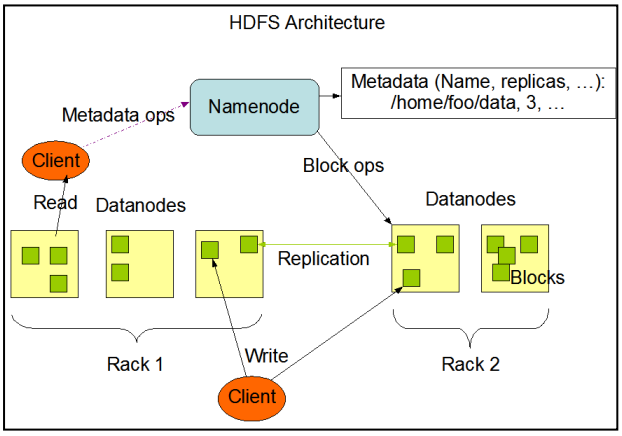
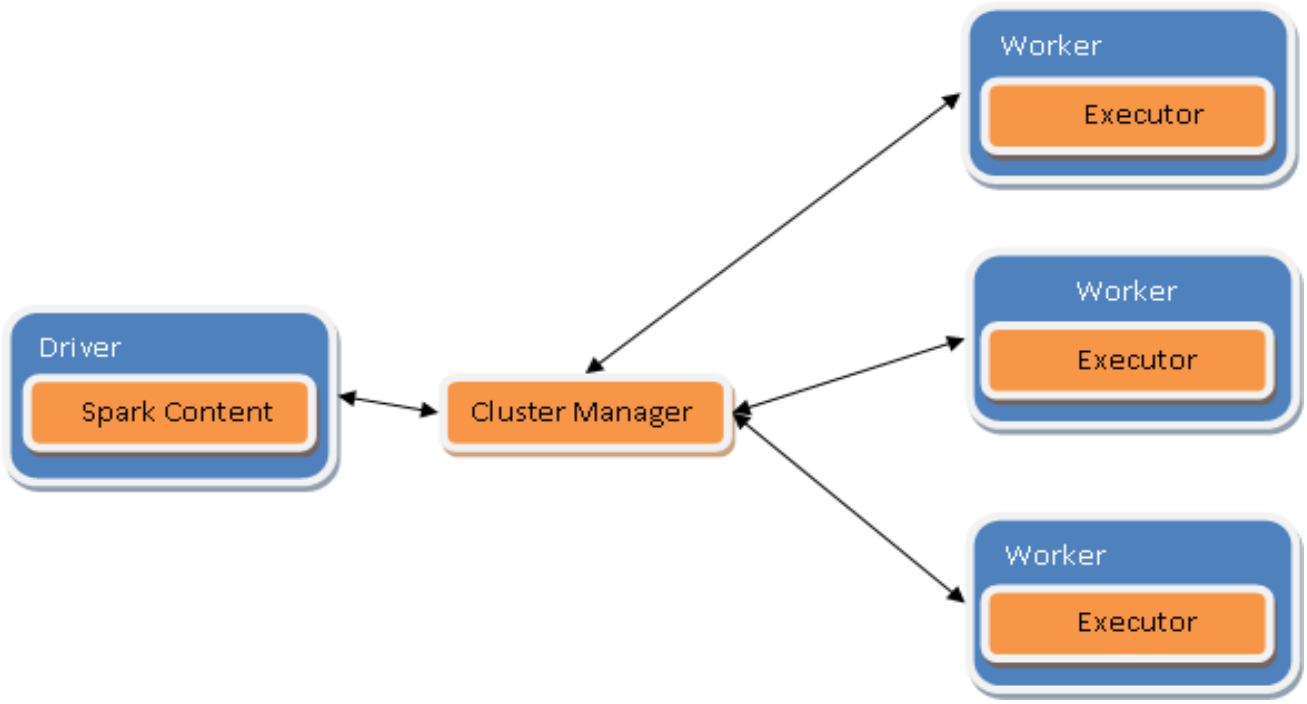


Figure 2: Architecture of HDFS

As presented in Figure 2, master/slave architecture of HDFS is provided. HDFS is associated with a cluster of nodes. Each cluster contains a master server known as NameNode. It is able to manage file system and helps clients to access files. There are many other nodes called as DataNodes that are part of cluster and manage storage. HDFS is distributed in nature and all nodes in cluster can access it. Data is managed in the form of files. However, data is divided into blocks and they are stored in DataNodes. File s



ystem operations are executed by NameNode. Mapping blocks to DataNodes is also done by NameNode. The operations performed by DataNodes include creation of blocks, deleting blocks and making replicas of blocks based on the instructions coming from NameNode. NameNode presence in a cluster simplifies the tasks. HDFS gives support for traditional file organisation also. Thus users can automatically create directories and sub-directories. HDFS is designed to deal with large volumes of data. Replication decisions are taken by NameNode. This node receives periodical Heartbeat and Blockreport from DataNodes. This periodical notification indicates that DataNode is functioning well.

2.2 Apache Spark

Apache Spark is known as a distributed computing framework that comes under a new generation. It was designed to improve iterative workloads and their performance with memory computations. For such workloads, Hadoop is found to be inefficient due to more number of disc access requirement. There are two main contributions made by Spark. The first contribution is that it has introduced a new data structure known as Resilient Distributed Dataset (RDD). It plays crucial role in memory computing. The second contribution is that it has proposed rich software architecture that caters to the needs of wide

variety of applications such as streaming, machine learning and graph processing. transformations are made in the last possible moment when users want to collect data. Each RDD is associated with sequence of operations that can be applied on the initial slice of data. Spark operations are the operations on RDDs. Each Spark operation works on a partition of RDD. The tasks may have relationship such a dependent or concurrent. The tasks are of two types known as collection and transformation. model which enriches RDD has immutable data that can be created initially from HDFS. Later it can be modified by subjecting to to be various user defined functions and help in creation of computation pipeline. RDDs are subjected to lazy computations. It does mean that The dependences are of two types again. They are called as transformations using collection methods. In fact, RDD helps in implementation of improved version of MapReduce narrow and wide. When there are two consecutive map() functions, it is known as narrow dependency. On the other hand, a wide dependency includes shuffle and data is move reduced. More details of Spark architecture can be found in Figure 4.

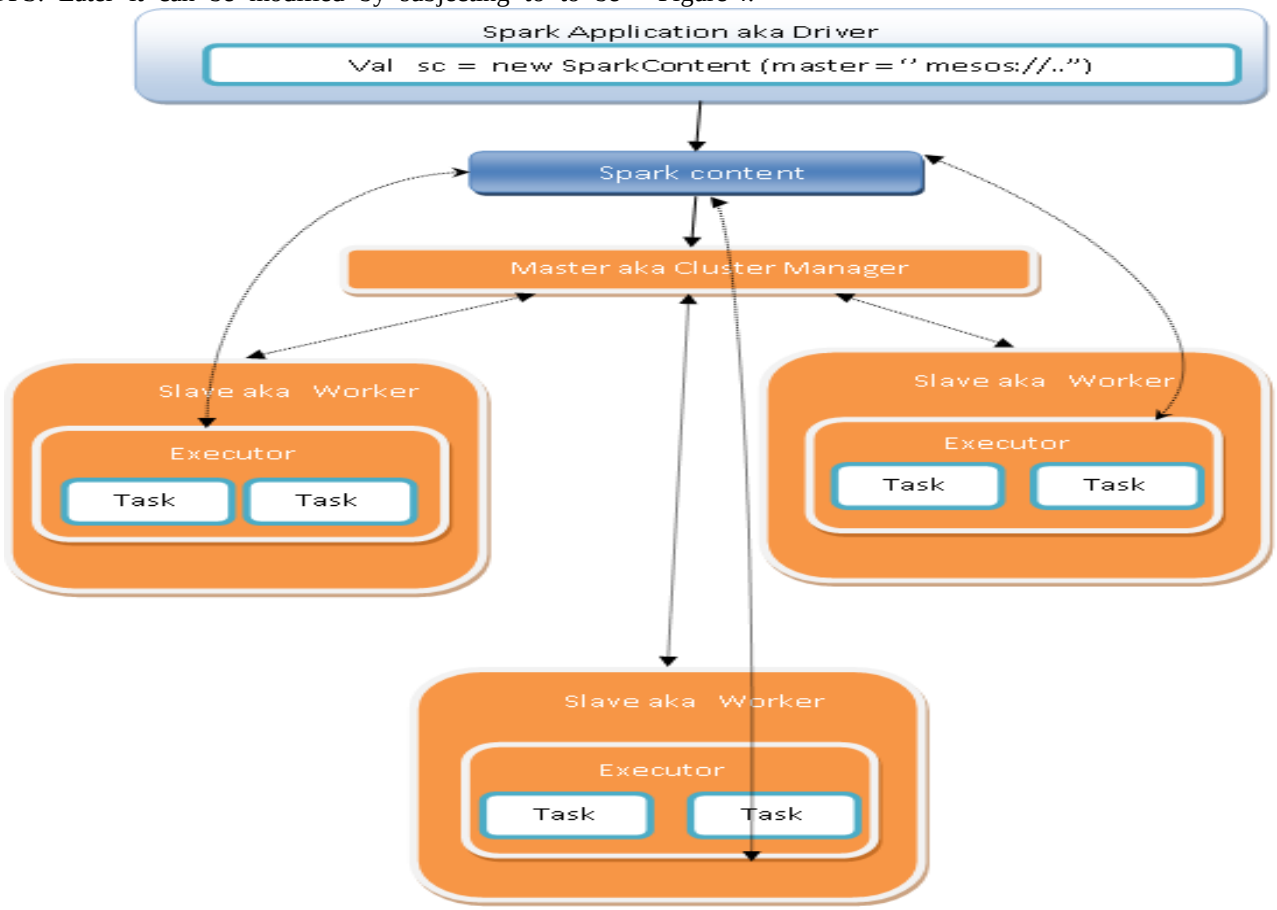


Figure 4: More details of Apache Spark

At the time of compilation, Spark analyses all operations associated with RDDs in a given application. Then it generates Directed Acyclic Graph (DAG). It is meant for representing both narrow and wide dependencies into two different stages. Tasks are assigned to worker nodes that hold data which is known as delay scheduling. When data is not in the worker, the workers offer to handle data may be denied. Data locality is configured at different levels. They are known as cluster, rack and node. Thus Spark will have increased tolerance with non-locality. In addition to this, the framework has ability to tolerate faults. RDDs can be instantiated and associated with different kinds of computations. Thus there is existence of columnar RDDs, file RDDs and graph RDDs.

2.3 Apache Flink

Apache Flink is another important distributed computing framework. It is meant for dealing with bounded and unbounded data streams with stateful computations. Flink is designed to run in any cluster environment and perform its operations with scalability and in-memory speed. There are some important aspects of the architecture of Flink.

Process Unbounded and Bounded Data

Data can be considered as a stream of events. For instance, user interactions on web applications or mobile applications, machines logs, sensor measurements and credit card transactions produce streams of data. Such data can be processed in the form of either bounded streams and unbounded streams. Unbounded streams do have a start but without a clearly defined end. They provide data continuously as it is generated and do not terminate. Therefore, such streams are to be processed continuously.

As the events occur or data arrive, that needs to be handled. Here it is not feasible to wait until all data is accumulated and then process it at once. Therefore, processing such data needs events that come in specific order to be processed in that order for completeness. Bounded streams on the other hand have both start and end points defined. Thus they can be processed after ingesting all data before starting computations. It also does not need any ordered ingestion. This kind of processing is also known as batch processing. Apache Flink is well known for processing both kinds of streams with high performance. It has precise control over time and state that is suitable for any kind of data. Bounded streams are actually process internally by algorithms that are designed to deal with fixed size data and provide excellent performance. Apache Flink is distributed in nature and needs computing resources in order to run applications. It can be integrated with cluster resources managers of Kubernetes, Apache Moses and Hadoop YARN. It can also be setup with a standalone cluster environment.

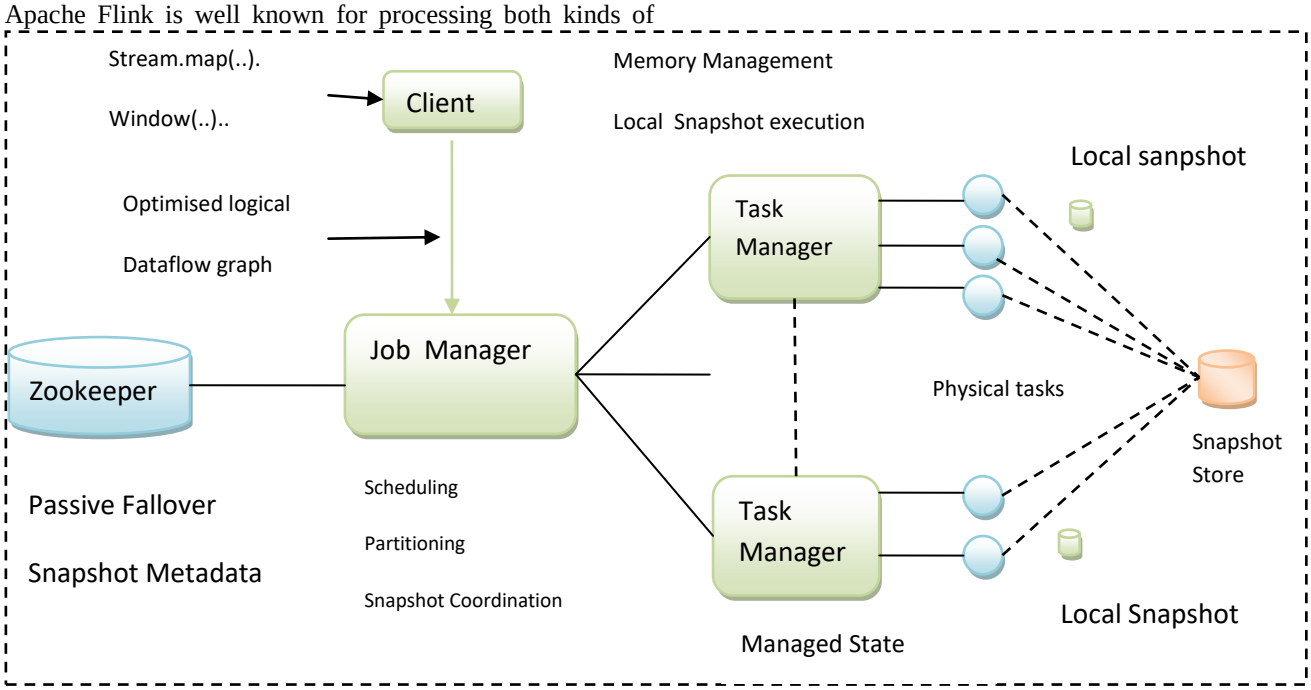


Figure 5 shows architectural overview of Apache Flink. It can work with any resource manager with the availability of resource-manager-specific models for deployments. Thus Flink can interact with any such manager in the way expected. After deploying a Flink applications, Flink can identify the resources required automatically. It is done based on the configuration provided in the application. It case of any failure, Flink is able to replace failed container with newly requested resources. It does all communication calls with REST technology. Thus Flink can be integrated with other distributed computing frameworks.

Scalability of Flink

Apache Flink is designed for execution of stateful streaming applications with scalability. Applications are executed in parallel by dividing into thousands of tasks that are assigned many worker machines in a given cluster. Thus an application can exploit main memory, network I/O and CPUs of any number of worker nodes. It can also maintain application state in scalable fashion. It also works with algorithms containing incremental checkpoints and asynchronous communications with minimal overhead on processing and ensures good latency performance. Users of Flink reported that it has high scalability and ability to process huge number of applications with voluminous data and run in thousands of machines.

Leverage In-Memory Performance

Applications that run in Flink with stateful feature are automatically optimised to have local state access. The state of tasks is maintained in memory. If the state size is more than available memory, then it can access on-disk data structures efficiently. Thus tasks in Flink can perform computations by using in-memory, accessing local and providing low processing latencies. It has a feature known as exactly-once state consistency that is guaranteed. This feature helps in case of failures as it considers asynchronous communication and checkpoints in the local state in order to have durable storage.

2.4 Apache Storm

Apache Storm is another open source distributed processing framework. It is for real time computations. Its batch processing is similar to that of Hadoop. Apache Stock can handle unbounded streams in a reliable fashion. It can process millions of jobs in very short span of time. It is also integrated with Hadoop for enhancingthroughputs. It can be integrated with any programing language and its applications are easier to implement. Nathan Marz developed Storm and later on it was taken by Apache Software Foundation in 2013. It was developed to have big data analytics easier.

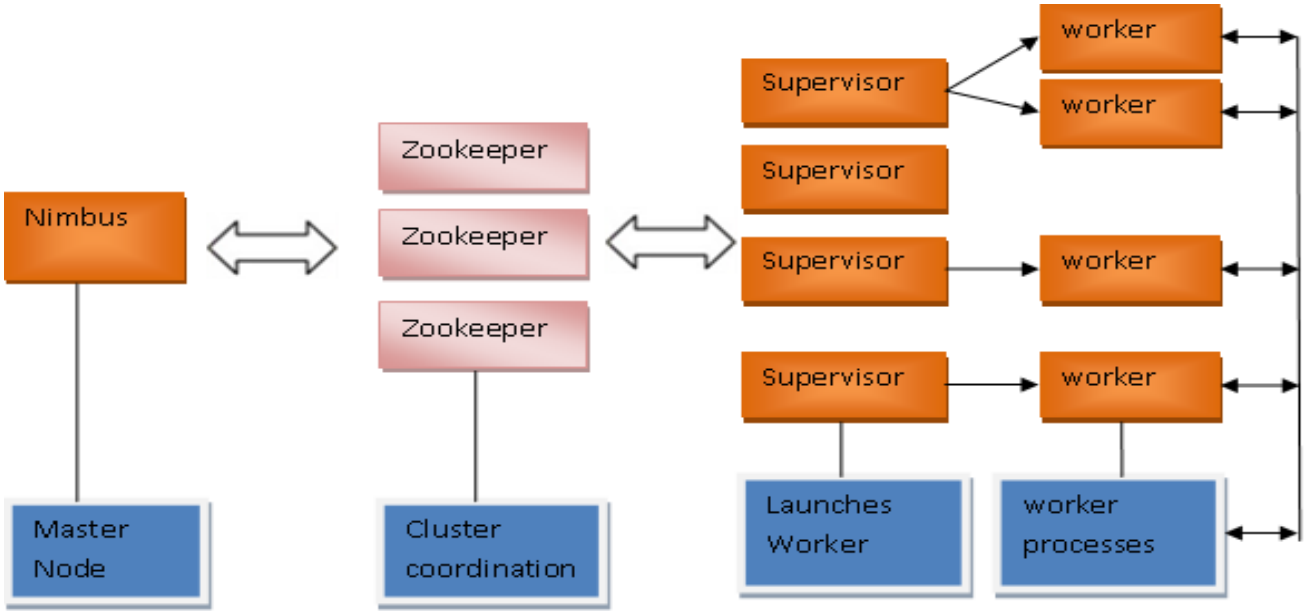


Figure 6: Overview of Apache Storm

As shown in Figure 6, the framework resembles Hadoop. However, there are significant differences as well. There are two kinds of nodes known as master node and worker node. Master nodes runs on Nimbus which is a daemon. It is similar to the Job Tracker in Hadoop. Nimbus is meant for assignment of tasks to worker nodes, distribution of codes and monitoring performance. The worker nodes on the other hand run on Supervisor daemon. It is meant for executing one or more worker processes. Nimbus assigns works to supervisor. Each worker process needs specific topology to run. It has no ability to manager cluster state. Instead, it depends on Apache Zookeeper to achieve the same. The Zookeeper helps in communication between the Nimbus and supervisors in the form of processing status messages and message acknowledgements.

2.5 Apache Samza

Apache Samza is another distributed stream processing framework developed at LinkedIn in 2013. Samza supports many real world use cases like big data analytics. It can also be integrated with Apache Kafka, Brooklin, ElasticSearch, HDFS, Amazon Kinesis and Azure EventHubs. It is being used by many companies like Optimizely, eBay, TripAdvisor and Slack. Apache Samza has rich set of API, integration with Samza SQL and Apache Beam, event-time based processing. It can also be run in standalone mode without YARN.

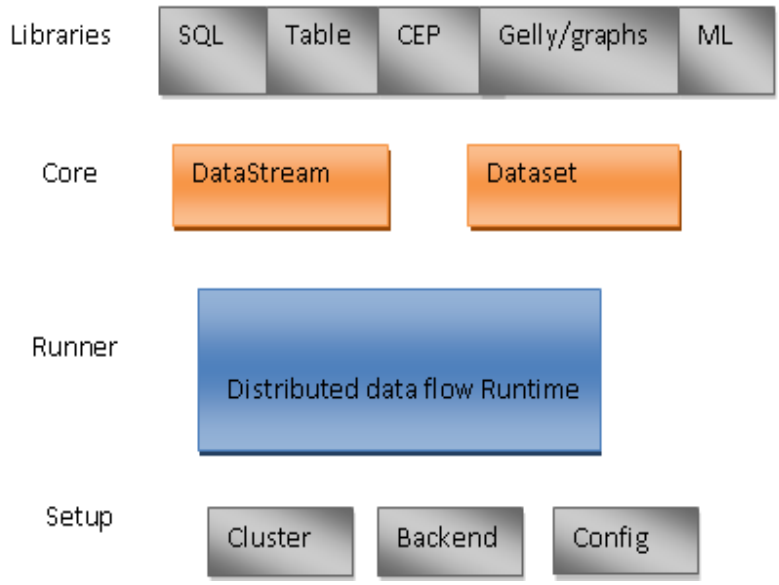


Figure 7: Overview of Apache Samza

As shown in Figure 7, Apache Samza was developed to solve many fundamental problems like scaling out processing, handling failures, managing local state, checkpoints and so on.

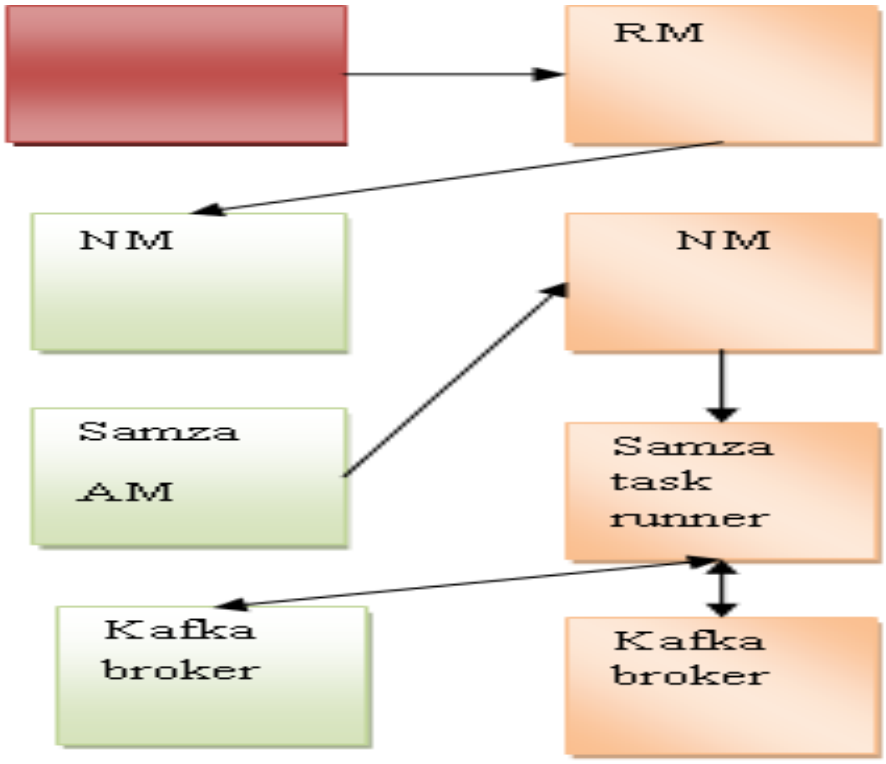


Figure 8: More details of Apache Samza

It was developed to be streaming-first execution engine in distributed environment which is made up of simple API. Then the applications are scaled by breaking into tasks. Each task can work on give portion of data with its own local storage. Each write operation is replicated into change log provided by Kafka. Thus it can recover from failures faster. It also has incremental checkpoints and supports large scale stateful processing. Thus it is meant for processing streams that need high level of performance.

III. EXPERIMENTAL RESULTS

Experiments are made on the big data processing using distributed programming frameworks. Two frameworks are used in our empirical study in this paper. They are known as Apache Flink and Apache Spark. Observations are made on processing big data and evaluating in terms of execution time,

3.1 Results on Execution Time

Execution time of the two frameworks is studied with TeraSort benchmark. This benchmark is used with different data size. The results are presented in Table 1.

Dataset Size	**Execution Time (seconds)**	
	Apache Flink	**Apache Spark**
100 MB	0	10
1 GB	10	30
10 GB	50	100
100 GB	700	1300

Table 1: Execution time recorded for both frameworks against different size of data

As presented in Table 1, the execution time in seconds is provided for Apache Flink and Apache Spark frameworks with TeraSort benchmark against different sizes of dataset (big data).

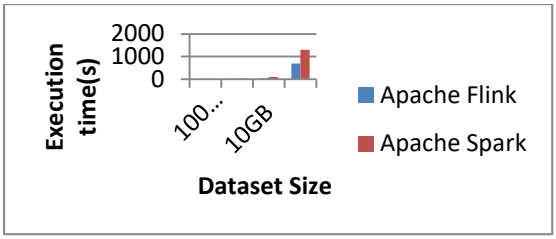


Figure 9: The execution time of Apache Flink and Apache Spark

As presented in Figure 9, it is evident that the dataset size with 100 MB, 1 GB, 10 GB and 100 GB is provided in horizontal axis. The vertical axis presents the time taken to execute that that with TeraSort benchmark.

The results revealed that the dataset size has its impact on the time taken. Another observation is that Apache Flink is faster than that of Apache Spark.

3.2 Results on Network Usage

This sub section provides results of network usage by the frameworks. The results of Apache Flink and Apache Spark are provided here in terms of outgoing and incoming traffic.

Time(s)	Network usage	
	Incoming traffic	Outgoing traffic
100	270	270
200	250	250
300	250	250
400	250	250
500	670	570
600	570	630
700	650	550
800	280	400

Table 2: Network usage performance of Apache Flink

As shown in Table 2, the network usage of Apache Flink is studied with observations made on incoming traffic and outgoing traffic. Elapsed time is considered for observations.

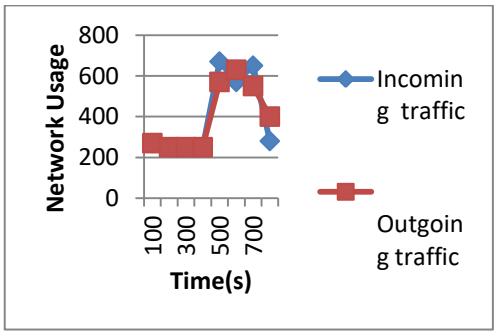


Figure 10: Network usage of the frameworks against elapsed time of Apache Flink

As shown in Figure 10, it is evident that the elapsed time is presented in horizontal axis while the network usage is presented in vertical axis. The incoming and outgoing traffic showed comparable trends. When the elapsed time is from 100 to 400 seconds, they showed same network usage. Later on they showed slightly different usage. However, there is no much difference between incoming traffic and outgoing traffic in terms of network usage.

Time(s)	Network usage	
	Incoming traffic	Outgoing traffic
200	95	95
400	0	0
600	120	120
800	620	540
1000	380	385
1200	95	140
1400	195	130
1600	50	60

Table 3: Network usage performance of Apache Spark

As shown in Table 2, the network usage of Apache Spark is studied with observations made on incoming traffic and outgoing traffic. Elapsed time is considered for observations.

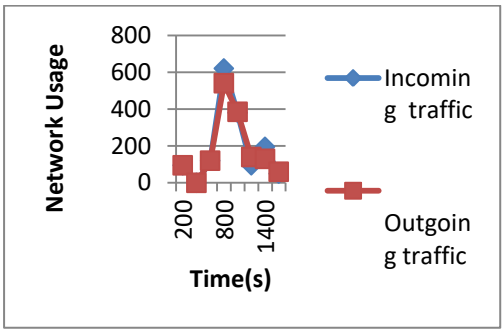


Figure 11: Network usage of the frameworks against elapsed time of Apache Spark

As shown in Figure 11, it is evident that the elapsed time is presented in horizontal axis while the network usage is presented in vertical axis. The incoming and outgoing traffic showed comparable trends. When the elapsed time is from 100 to 600 seconds, they showed same network usage. Later on they showed slightly different usage. However, there is no much difference between incoming traffic and outgoing traffic in terms of network usage. These observations are made with TeraSort benchmark.

3.3 Results on Throughput

This sub section provides results of empirical study on the two frameworks in terms of throughput in terms of read and write operations

Table 4: Throughput performance of Apache Flink

. Time(s)	Throughput	
	Read	Write
100	170	220
200	250	250
300	250	250
400	250	250
500	380	950
600	370	980
700	360	700
800	200	950
900	50	280

As shown in Table 4, the throughput performance of Apache Flink is studied with observations made on read and write operations. Elapsed time is considered for observations.

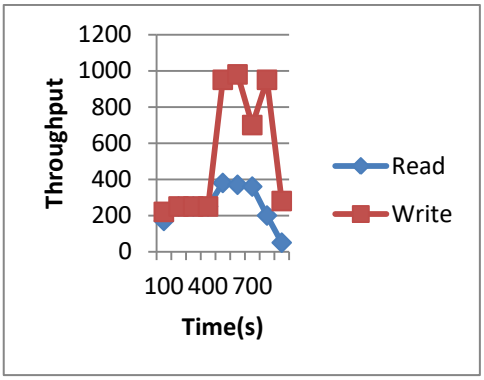


Figure 12: Throughput performance of Apache Flink against elapsed time

As shown in Figure 12, it is evident that the elapsed time is presented in horizontal axis while the throughput of Apache Flink is presented in vertical axis. The read and write operations showed comparable trends. When the elapsed time is from 100 to 400 seconds, they showed same throughput. Later on they showed different throughput. It is observed that write operation throughput is much higher than that of read operation. These observations are made with TeraSort benchmark.

Time(s)	Throughput	
	Read	Write
200	190	190
400	660	380
600	140	120
800	180	750
1000	190	660
1200	150	280
1400	140	190
1600	130	250
1800	120	160

Table 5: Throughput performance of Apache Spark

As shown in Table 5, the throughput performance of Apache Spark is studied with observations made on read and write operations. Elapsed time is considered for observations.

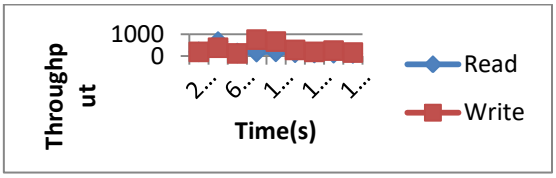


Figure 13: Throughput performance of Apache Flink against elapsed time

As shown in Figure 13, it is evident that the elapsed time is presented in horizontal axis while the throughput of Apache Spark is presented in vertical axis. The read and write operations showed comparable trends. Unlike the throughput of Apache Flink, they showed different throughput throughout the elapsed time. It is observed that write operation throughput is much higher than that of read operation after reaching 600 seconds elapsed time. These observations are made with TeraSort benchmark.

IV. CONCLUSIONS AND FUTURE WORK

This paper studied various distributed programming frameworks that are associated with cloud computing platforms. These frameworks are basically for storing, processing and retrieving big data. They are associated with Distributed File Systems (DFSs). The frameworks studied include Hadoop, Apache Samza, Storm, Apache Flink and Apache Spark. The architectural overview of the frameworks is provided besides functioning of them. Two important frameworks like Apache Flink and Apache Spark are used for empirical study. They are evaluated with TeraSort benchmark in terms of execution time, network usage and throughput. From the experiments it is understood that Apache Flink is better than that of Apache Spark for processing big data. These observations are based on the datasets with different size. However, this study is to be continued (out future work) for more datasets of very large size from different domains in the real world. Unless such evaluation is made it is not possible to have generalized conclusions. However, it is understood that Flink is faster than that of Spark. Another direction for future work is to study security issues associated with cloud computing with respect to big data analytics and these distributed programming frameworks.

REFERENCES

1. Paris Carbone, Stephan Ewen, Gyula Fora, SeifHaridi, Stefan Richter, Kostas Tzoumas. (2017). State Management in Apache Flink, 10 (12), p1-12.
2. ShelanPerera, AshansaPerera, Kamal Hakimzadeh. (2016). Reproducible Experiments for Comparing Apache Flink and Apache Spark on Public Clouds, p1-10.
3. Do Le Quoc, Ruichuan Chen, PramodBhatotia. (2017). Approximate Stream Analytics in Apache Flink and Apache Spark Streaming, p1-14.
4. IlyaVerbitskiy, LauritzThamsen, Odej Kao. (2016). When to Use a Distributed Dataflow Engine: Evaluating the Performance of Apache Flink, p1-8.
5. K.M.J. Jacobs. (2015). Apache Flink: Distributed Stream Data Processing, p1-5.
6. Martin Kleppmann,Jay Kreps. (2015). Kafka, Samza and the Unix Philosophy of Distributed Data. *IEEE*, p1-11.
7. Shadi A. Noghabi, KartikParamasivam , Yi Pan , Navina Ramesh, Jon Bringhurst. (2017). Samza: Stateful Scalable Stream Processing at LinkedIn10 (12), p1-12.
8. Massimo Ficco , Roberto Pietrantuono, Stefano Russo. (2017). Aging-related performance anomalies in the apache storm stream processing system. *Elsevier*, p1-21.
9. Marcos Dias de Assuncao, Alexandre Da Silva Veith, RajkumarBuyya. (2017). Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions. *Elsevier*, p1-25.

10. Muhammad HussainIqbal , Tariq Rahim Soomro . (2015). Big Data Analysis: Apache Storm Perspective. *International Journal of Computer Trends and Technology (IJCTT)*. 19 (1), p1-7.
11. P. Taylor Goetz, Brian O'Neill. (2014). Storm Blueprints: Patterns for Distributed Real-time Computation, p1-336.
12. Suresh Marru,LahiruGunathilake,ChathuraHerath,PatanachaiTangchaisin,MarlonPierce,ChrisMattmann,RaminderSingh,ThilinaGunarathne,EranChinthaka,,Ross Gardler,AleksanderSlominski,AteDouma. (2012). Apache Airavata: A framework for Distributed Applications and Computational Workflows. *ACM*, p1-8.
13. MateiZaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. (2010). Spark: Cluster Computing with Working Sets, p1-7.
14. Feng Zhang, Min Liu, Feng Gui, WeimingShen, AbdallahShami, Yunlong Ma. (2015). A distributed frequent itemset mining algorithm using Spark for Big Data analytics. *Springer*. 18, p1493-1501.
15. SasmitaPanigrahia, RakeshKu,Lenkaa , AnanyaStitipragyana. (2016). A Hybrid Distributed Collaborative Filtering Recommender Engine Using Apache Spark. *Elsevier*. 83, p1000-1006.
16. Telmo da Silva Morais. (2015). Survey on Frameworks for Distributed Computing: Hadoop, Spark and Storm, p1-11.
17. Eric Jonas, QifanPu, ShivaramVenkataraman, Ion Stoica, Benjamin Recht. (2017). Occupy the Cloud: Distributed Computing for the 99%, p1-8.
18. Sayantan Sur, Hao Wang, Jian Huang, Xiangyong Ouyang and Dhabaleswar K. Panda. (2012). Can High-Performance Interconnects Benefit Hadoop Distributed File System, p1-10.
19. SairamGurajada, Stephan Seufert, Iris Miliaraki, Martin Theobald,. (2014). TriAD: A Distributed Shared-Nothing RDF Engine based on Asynchronous Message Passing, p1-12.
20. Jeremy Kepner, William Arcand, William Bergeron, Nadya Bliss, Robert Bond, ChansupByun, Gary Condon, Kenneth Gregson, Matthew Hubbell, Jonathan Kurz, Andrew McCabe, Peter Michaleas, Andrew Prout, Alb. (2013). Dynamic Distributed Dimensional Data Model (D4m) Database And Computation System, p1-4.
21. Myoungjin Kim, Seungho Han, Yun Cui ,Hanku Lee and ChangsungJeong. (2012). A Hadoop-based Multimedia Transcoding System for Processing Social Media in the PaaS Platform of Smccse . *Ksii Transactions On Internet And Information Systems*. 6 (11), p1-12.