ISSN 2007-9737

Performance Analysis of Distributed Computing Frameworks for Big Data Analytics: Hadoop Vs Spark

Shwet Ketu[1], Pramod Kumar Mishra[1], Sonali Agarwal[2]

[1] Banaras Hindu University, Institute of Science,
Department of Computer Science,
India,

[2] Indian Institute of Information and Technology, Allahabad,
India

shwetiiita@gmail.com, mishra@bhu.ac.in, sonali@iiita.ac.in

Abstract. In the last one decade, the tremendous growth in data emphasizes big data storage and management issues with the highest priorities. For providing better support to software developers for dealing with big data problems, new programming platforms are continuously developing and Hadoop MapReduce is a big game-changer followed by Spark, which sets the world of big data on fire with its processing speed and comfortable APIs. Hadoop framework emerged as a leading tool based on the MapReduce programming model with a distributed file system. Spark is on the other hand, recently developed big data analysis and management framework used to explore unlimited underlying features of Big Data. In this research work, a comparative analysis of Hadoop MapReduce and Spark has been presented based on working principle, performance, cost, ease of use, compatibility, data processing, failure tolerance, and security. Experimental analysis has been performed to observe the performance of Hadoop MapReduce and Spark for establishing their suitability under different constraints of the distributed computing environment.

Keywords. Big data, parallel processing, distributed environments, distributed frameworks, Hadoop MapReduce, Spark, big data analytics.

1 Introduction

Recently, tremendous growth is seen in Information Technology (IT) applications where data in terms of various sizes, velocities, and veracities have been observed that give the new field of research known as big data. Big Data scenario motivates re-searchers to work on high-speed data processing and management schemes which are not possible by using traditional approaches and demands distributed processing infrastructure [1, 2, 3].

Hadoop, a framework of a distributed processing environment is designed and developed for batch processing based on the MapReduce programming model. It runs in a distributed manner on various systems and can be run as well in a standalone mode with full potentiality and functionality. It was developed for handling the huge amount of data across various systems in less time with great performance [4].

Spark is also a programming framework just like Hadoop but it is designed and developed for real-time along with batch processing. It works in a fully distributed fashion and more powerful to handle the big data problems as compared to Hadoop MapReduce with the help of its streaming API, which pro-vides continues the processing of data in terms of short interval batches. In Spark, jobs may run continuously till shutdown either by users or by any unrecoverable failure. It can also run in a standalone environment [5].

In this paper, we have performed a comparative study of distributed frameworks and find out its impact over big data. Spark and Hadoop MapReduce frameworks are being used for this purpose. Both are capable of big data processing in a distributed manner using large datasets.

The paper is organized as follows: In section II, the state of art in the field of the distributed framework is discussed.

A comparative study is grounded on the working principle of Spark and Hadoop MapReduce is discussed in section III. In section IV, analysis of these frameworks based on various parameters are performed and discussed. Section V shows the performance evaluation, which is observed using the standalone setup of both distributed frame-works by using two benchmark datasets. Concluding comments with future perspective has been presented in the last section VI.

2 State of Art

In this section, the working principle of Spark and Hadoop MapReduce has been discussed in detail, as follows:

2.1 The Hadoop MapReduce Approach

Hadoop is a MapReduce based distributed framework, designed to process the enormous volume of data on several nodes or computers. It is based on the concept of parallel data processing that makes it cost and time effectively. It is based on the MapReduce programming model [6]. The two pillars of Hadoop are:

2.1.1 YARN

YARN represents *Yet Another Resource Negotiator*. It is aimed to provide more features for those applications, which are running in a Hadoop distributed cluster environment. The key functionalities are Memory organization, which assigns CPU and storage support to the applications that are running on Hadoop distributed environment. In the initial versions, YARN was absent due to the fact that only MapReduce jobs were implemented, but in a more recent version, the presence of YARN allows the processing of other frameworks media to run on the Hadoop distributed environment [7, 8].

2.1.2 HDFS

HDFS represents the Hadoop Distributed File System, which empowers the system to perform the task in a distributed manner.

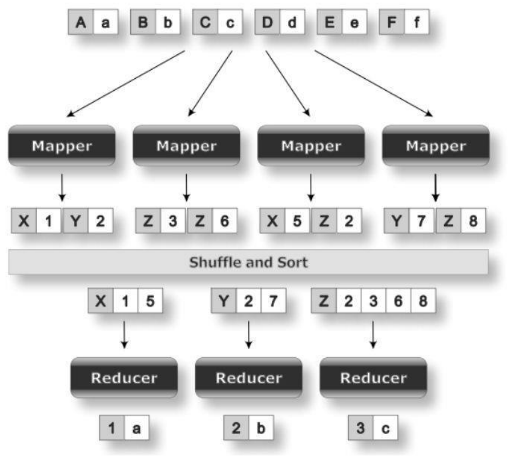


Fig. 1. Functioning of map and reduce programming model

Its works on the principle of by linking of all file systems, which are presented at several nodes, and merge them into a single file system [8, 9].

2.1.3 MapReduce

A job in Hadoop is made up of mainly two important functions, a map function and a reduce function. There are many other functions also available in Hadoop, which generally perform after the map function or in between the map and reduce functions. The initial task of the map function is to read the chunk of data from HDFS and return into (key, value) pairs with their distinct partitions.

For example, in-text mining, it works as a parser and cleaner for the dataset as shown in figure 1 where the input is partitioned in various mappers followed by reducer task in order to perform related computations in parallel. It is indicated in figure 1 that the output coming from the map stage is fed into the various reducers. The reducers are now read the input in (key, value) pairs and perform the computations on it like sum, average, etc. [10, 11].

The workflow of nodes in Hadoop is shown in figure 2.

All MapReduce task is divided into three nodes, job submission node, Namenode and Slave node for successful completion of Hadoop task. Job tracker and task tracker are used to tracking the MapReduce task and all the operation is performed

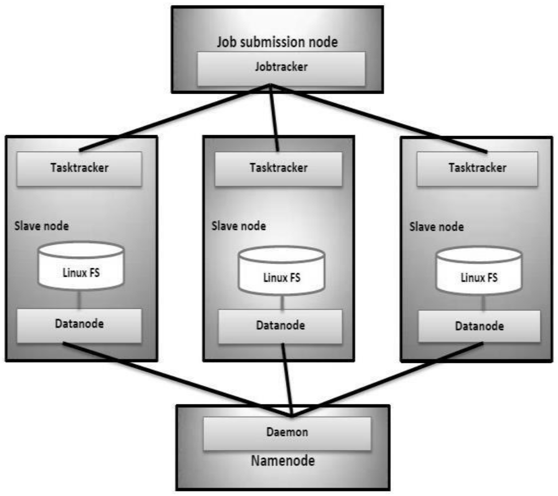


Fig. 2. Nodes workflow in MapReduce programming model

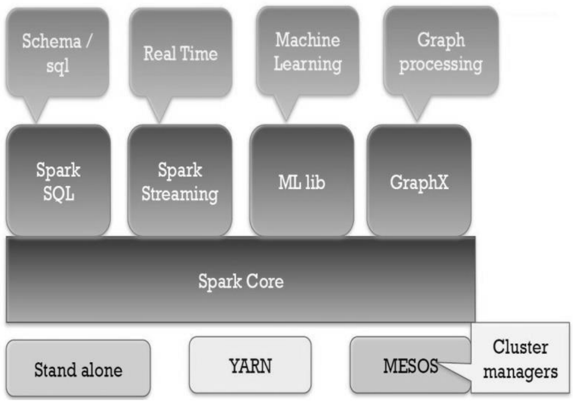


Fig. 3. Spark framework

on Datanode, controlled by Namenode. These operations are executed in the Slave node.

2.2 The Spark Approach

Spark is one of the big data analysis frameworks based on the concept of distributed processing. It was designed for processing the data in real-time.

It follows In-memory computing by which the processing speed becomes much faster as compared to disk-based computing in Hadoop. Spark has three language supports, which means, it does not follow only one programming language or model as Hadoop. By having the support of Python, Java, and Scala, it contains algorithms that can be developed using these programming languages. All the programing language, which is supported by Spark, is encapsulated with high-level APIs enables execution of graph too [11].

In addition to this, MLlib supports machine learning, Spark SQL is available for handling structured data and for handling graph, GraphX is used.

Parallel application development is possible using Spark Streaming. The main objective of Spark is to perform the task in a distributed either for real-time data or for batch data with less time obligation [12].

RDDs, which stand for Resilient Distributed Datasets, are supported by Spark. The data is collected from internal storage (HDFS), external storage, or a derived dataset formed by other RDDs. These RDDs are maintained by inbuilt options such as On-disk storage, Serialized data (In-memory storage) and Desterilized java objects (In-memory storage) [13, 14, 15].

In figure 3, an overview of the Spark framework is shown along with its collaboration to higher-level tools and APIs.

3 Discussion based on Working Principle

3.1 Working of Map Side Shuffle

Working of Map side shuffle phase of Hadoop MapReduce and Spark is shown in figure 4 [16, 17].

3.1.1 Map Side of Hadoop MapReduce

Step 1: At each map task the output of data come in (Key, Value) pair.

Step 2: The output is being stored in a buffer (circular) rather than writing directly to disk.

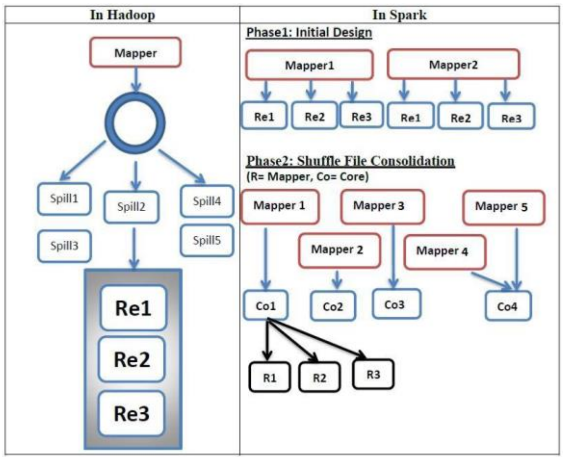


Fig. 4. Map Side: Shuffle phase differences (Hadoop vs. Spark)

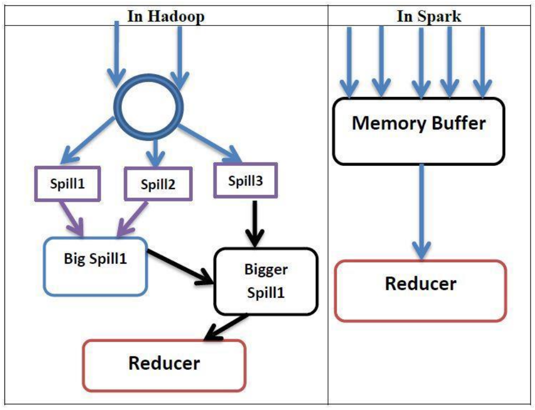


Fig. 5. Reduce side: Shuffle phase differences (Hadoop vs .Spark)

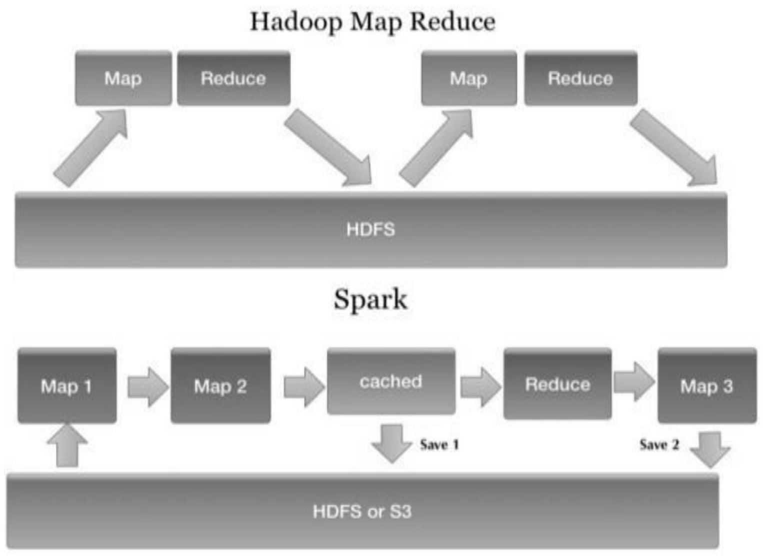


Fig. 6. Programming model workflow of Hadoop and Spark

Step 3: Circular buffer size is approximate 100MB and if the buffer size = 80% than the data is spilled. This process is called shuffling of spill files.

Step 4: Many map tasks produces many spill files on a specific node. Spill files of single map task (not all map tasks) are merged as one file, which is organized (sorted), and partitioning is done based on the total number of reducer pre-sent.

3.1.2 Map side of Spark-Initial Design

3.1.2.1 Initial Design Phase

Step 1: The output is written to an operating system buffer cache.

Step 2: It will depend upon the operating system whether the data will remain on the buffer or it will be spilled to disk.

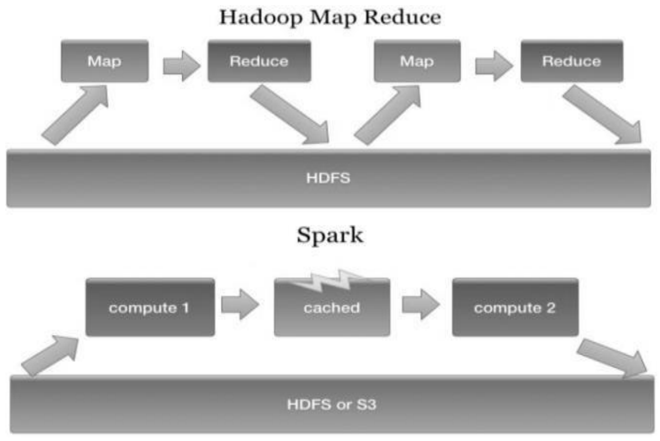


Fig. 7. Iterative workload handling in Hadoop and Spark

Table 1. Comparison based on performance evaluation

	Hadoop (MapReduce)	**Spark**
Data Processing	MapReduce persists back to the disk after a map or reduce action.	In-memory (all data and computation are performed and the result is loaded at the end)
Memory	Kill the process as soon as a job is done and release the memory.	A lot of memory needed (like standard DBs) and keep it safe there until further notice.
Running	Processes run alongside other services too.	Processes run alone.
Iterative Computations	It was designed for one-pass ETL-like jobs.	Give the best result in iterative, the same data many times.

Step 3: Shuffle spill files are created is equal to the number of reducers in each map task. It does not marge spills files into one big file.

3.1.2.2 Shuffle Files Consolidation

Step 1: The output is written to an operating system buffer cache. It will depend upon the operating system whether the data will remain on the buffer or it will be spilled to disk.

Step 2: Shuffle spill files are created as many as the number of reducers in each map task. It does not marge spills files into one big file.

Step 3: The map task is consolidated into a single file, which runs on the same cores. Therefore, the output generated by each core is as many shuffle files as the number of reducers.

3.2 Working of Reduce side Shuffle Phase

Working of Reduce side shuffle phase of Hadoop MapReduce and Spark is shown in figure 5 [16,17].

3.2.1 Reduce Side of Hadoop MapReduce

Step 1: The intermediate files, which are generated at the map side are being PUSHED and loaded into the memory.

Step 2: The data will spill to disk (if buffer = 70%)

Step 3: After that, the bigger files form from merging the spills.

Table 2. Comparison based on ease of use

	Hadoop (MapReduce)	**Spark**
APIs Support	Yes (only Java)	Yes (Java, Scala, and Python)
Language Support	Java only	Java, Scala, and Python
SQL Compatibility	Yes (Hive) [23]	Yes [Spark SQL (Shark)][24]
Programming model	Within MapReduce programming model	Within user define function
Inactive mode	No (command line)	Yes

Table 3. Comparison based on cost evaluation (on single node)

	Hadoop (MapReduce)	**Spark**
Open-source	Yes	Yes
Run-on cloud	Yes	Yes
Cores	4	8-16
Memory	24GB	8GB
Disks	4-6 one-TB disks	4-8
Network	1 GB Ethernet all-to-all	10GB or more

Table 4. Comparison based on compatibility measurement

	Hadoop(MapReduce)	**Spark**
Support of data sources of Hadoop	Yes	Yes
Support file formats	Yes	Yes
Work with BI tools	No	Yes
Code size (Lines of code)	Very Long (due to only java support)	Less (due to python and scala support)

Step 4: At the end, the reduce method gets invoked.

3.2.2 Reduce Side of Spark

Step 1: The intermediate files being PULLED to Reducer.

Step 2: In this step, the data is written to memory.

Step 3: If the data size is large and it does not fit in the memory, Out of Memory Exception (OOM) occurs but it is being solved in 0.9 and its grater version via spilling the data to disk.

Step 4: At the end, the reduce functionality gets invoked.

Table 5. Comparison based on data processing

	Hadoop (MapReduce)	**Spark**
Data processing	Batch (apache mahout)	Real-time and batch(Storm or Impala and Giraph)
Machine learning libraries	No	Yes

Table 6. Comparison based on failure tolerance

	Hadoop (MapReduce)	**Spark**
Retires per task	Yes	Yes
Speculative execution	Yes	Yes
Relies on	Hard drives	Buffers
Time is taken (Cost)	Less	More

Table 7. Comparison based on security

	Hadoop (MapReduce)	**Spark**
Security via	Use Hadoop's security policy	Sheared secret key and Kerberos
Security feature vise	Highly reliable	Less reliable

Table 8. Comparison in terms of machine learning support

	Hadoop (MapReduce)	**Spark**
Tools Support for machine Learning	Mahout [29]	MLLib
APIs Support For	Java only	Java, Scala, and Python
Iterative algorithm performance (processing speed)	Slower	Very fast (due to In-memory computation)
Support In-memory processing	Yes	No

Based on the working principle it is clear that both frameworks work in a different style but also have some common functionality.

3.3 Programming Model of HADOOP and SPARK

The workflow of Spark and Hadoop MapReduce programming models are shown in figure 6. MapReduce based programming model is less generic than Spark because Spark is based on In-memory computation model, which makes Spark more effective, powerful and easy to use. The Hadoop work on the principle On-disk MapReduce programming model that means the data is stored on secondary storage. The Hadoop Distributed file system (HDFS) is the place where the data is stored and where all computation is being performed. Due to On-disk computation, the processing turns out to be slow in Hadoop, which

Table 9. Comparison in terms of ETL support

	Hadoop (MapReduce)	Spark
ETL tools support	Yes (Pig, Cascading, Oozie) [21]	Yes (Native RDD programming {Scala, Java, Python}) [21]
ETL Workflow	High-level ETL workflow	Spork is used
Cascading level	High level	Support only Spark- scalding

Table 10. Comparison in terms of SQL support

	Hadoop (MapReduce)	Spark
Engine	Hive [23]	Spark SQL [25]
Language	HiveQL	HiveQL and RDD programming language (Java, Scala, Python)
Scale	Petabytes	Terabytes
Inter-operability possible	No	Yes (Can read Hive tables or standalone data)
Formats	CSV, JSON, Parquet	CSV, JSON, Parquet

consequently increases processing delay and time complexity. Hadoop having the support of Java language and its linked APIs but Spark have to support of three programming languages i.e. Scala, Java and Python with its linked APIs [17, 18, 19].

3.4 Better Fit for Iterative Workloads Handling

Figure 7 shows the iterative workload handling models for Hadoop MapReduce and Spark For iterative workload handling, Spark is much better than Hadoop because it follows In-memory and cache-based processing. Cache-based processing enables Spark to work efficiently and faster for iterative algorithms. It follows In-memory based computation by which fetching of data from local cache reduces the computation time and data loading time.

While in Hadoop MapReduce, due to it is on disk computation through HDFS, it is not possible. Hence, it is clear that for iterative computation, Spark is more powerful and faster than Hadoop MapReduce [20].

Both the framework has its own advantages that establish its completeness and usability for handling the big data processing.

4 Comparison of Hadoop and Spark

A comparative analysis based on resource utilization and application scope of Hadoop MapReduce and Spark have been performed to highlight their similarities and dissimilarities [21].

4.1 Analysis based on Resource Utilization (Hadoop vs Spark)

4.1.1 Performance

From the table 1, it is clear that when all data fits in the memory (buffer memory) the performance of the Spark is better and when the data does not fits in the memory and it is needed to run other services along with the main process, the performance of the Hadoop MapReduce would be excellent [22].

Table 11. Comparison in Terms of Streaming Support

	Hadoop (MapReduce)	**Spark**
Stream Processing / Real-Time processing	Storm [27] Kafka [28]	Spark Streaming [26]
Processing of live data streams possible	No	Yes
Streaming API support	For Java only	Java, Scala, and Python
Real-time lookups	NoSQL (Hbase, Cassandra...etc.)	No Spark component. But Spark can query data in NoSQL stores

4.1.2 Ease of Use

In table 2, it is clearly mentioned that Spark is easier in context of programming and its interactive mode gives the smooth drive to the users as compared to Hadoop, which is quite difficult to work in the context of programming and handling due to its command line mode.

4.1.3 Cost

From table 3, it is clear that the Spark is more cost-effective than Hadoop MapReduce according to benchmarks. In respect to memory, the Hadoop require more due to its On-disk computation rather than Spark.

4.1.4 Compatibility

Both Spark and Hadoop MapReduce are the same in reference to data types and data sources, which are mentioned in table 4.

4.1.5 Data Processing

Hadoop is developed for batch processing but for real-time processing, Hadoop requires additional platforms. Spark is developed for both batch and real-time processing and there is no need for external platforms. Table 5 also indicates that machine learning libraries are also missing in Hadoop MapReduce.

4.1.6 Failure tolerance

From table 6, it is clear that the Spark and Hadoop MapReduce both retires per task and follows speculative execution. However, MapReduce starts on the hard drives, which give an advantage to Hadoop over Spark.

In Hadoop, if the process crashes in the middle of its execution, it restores current execution status next time. However, in Spark, this feature is absent and it starts the process from the very beginning due to that time complexity increases.

4.1.7 Analysis based on Resource Utilization (Hadoop vs Spark)

In the context of big data handling, the Spark is less secure than Hadoop. Hadoop MapReduce pro-jects are considered more reliable than Spark as shown in table 7.

4.2 Analysis based on Application Scope (Hadoop vs Spark)

4.2.1 Machine Learning Support

Table 8 shows the machine learning a feature-based comparison of both the frameworks for batter understating and suitability in Big Data analytics. Both frameworks having machine learning tools but Hadoop has only Java API support and it runs slowly with the iterative algorithm in turns of processing speed because it does not follow In-memory computation.

Table 12. A quick lookup of what has whatnot

Functionalities	Hadoop (MapReduce)	Spark
Operational Principle	Distributed Compute + Distributed Storage	Only Distributed Compute
Computation type	Based on MapReduce	Widespread computation
Storage area	Typically, data on HDFS (On-disk)	Both In-memory and On-disk
Iterative support	Not ideal for iterative workload	Perform Efficiently at Iterative workloads
Coding support	Java supported, Compact code	Java, Scala, and Python supported, Compact code
Process support	Batch process only	Batch and Real-time process both (work Up to 100x faster in In-memory and 2x - 10x faster on On-disk)
Hadoop and YARN capability	Yes	Yes
The flexibility of the programming model	Less flexible	More flexible
Data size Compatibility	Terabytes	Gigabytes
Usability	Relatively complex	Easy as compare to Hadoop
Great performance with	Large datasets (>= 100GB)	Small and medium datasets (<= 100GB)
Multiple APIs support	No (Java only)	Yes (Java, Scala, Python)
Distributed Storage	Use HDFS	Enables Cloud storage such as NFS mounts Or Amazon S3
SQL querying	By Hive	By Spark SQL
Machine Learning tools	Mahout [29]	MLLib [21]
NoSQL DB	Yes (Hbase)	No

4.2.3 SQL Support

Table 10 shows the SQL support-based comparison of both the frameworks for better understating, suitability and richness in terms of features. Both frameworks having SQL supporting tools but Hadoop SQL tool Hive works slow and having fewer features as compared to Spark. In terms of scalability, the Spark has terabytes support, which is very less than the Hadoop [25].

Table 13. Spark Key features

Key features	Explanation
Ease of Growth	✓ Having the support of Scala, Python and Java language ✓ Having quick and easy APIs support
Performance (In-memory)	✓ It follows the RDDs file system ✓ It enables DAGs Unify Processing
Joint Workflow	✓ Shark, Streaming, ML and GraphX support is enabled here

Table 14. Hadoop MapReduce Key features

Key features (Hadoop MapReduce)	Explanation
Unlimited Scalability	✓ For all users ✓ For all applications ✓ For all Data sources
Enterprise Platform	✓ More Reliable ✓ Healthier Security ✓ Offer Multi-tenancy
Wide Range of Applications	✓ Files ✓ Database ✓ Semi-structured

4.2.4 Streaming Support

In table 11 a comparison based on streaming support is discussed. The Spark supports its own Spark streaming but Hadoop uses the Strom and Kafka for stream and real-time processing respectively. For live data streams, the processing is possible only in Spark. In context to APIs, for streaming Hadoop has java support and, on another hand, Spark has Java, Scala, and Python support. For real-time lookup, NoSQL is used in Hadoop.

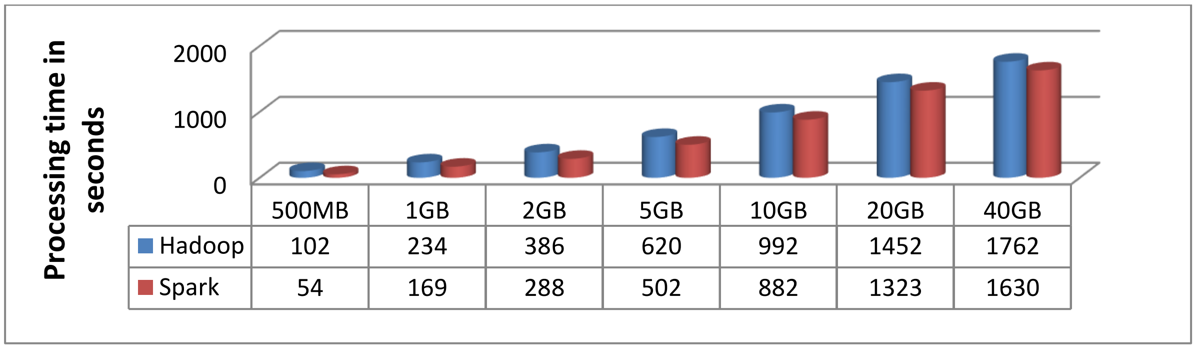
A quick evaluation of distributed programming framework on the basis of key features, functionally and use cases is achieved and deliberated in below table 12.

Key feature-based analysis of Spark and Hadoop is shown in table 13 and table 14 respectively.

The table demonstrations that Hadoop has a broader range of business platforms, better scalability, and broader application space, while Spark supports three languages, easy to adapt, and has outstanding computational abilities.

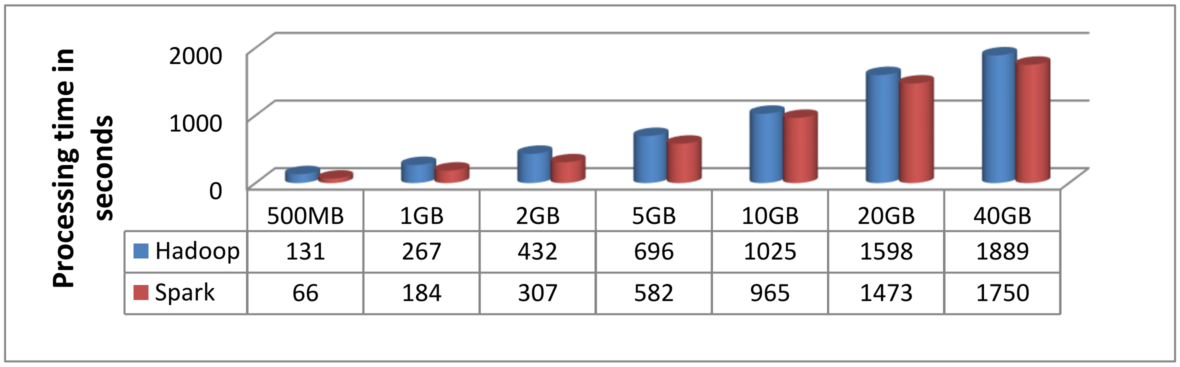
5 Experimental Evaluation

The experimental analysis is based on mainly three algorithms such as world count, logistic regression and distributed K-Means clustering. All experiments have been performed using four benchmark datasets which are: Wikilinks [30], Wikilanguage [30], Wikipedia [32] and Enron [31] dataset in Pseudo-Distributed Mode. The experimental settings and hardware required to perform the experiments on a single node (Pseudo Distributed Mode) are given in Table 14.



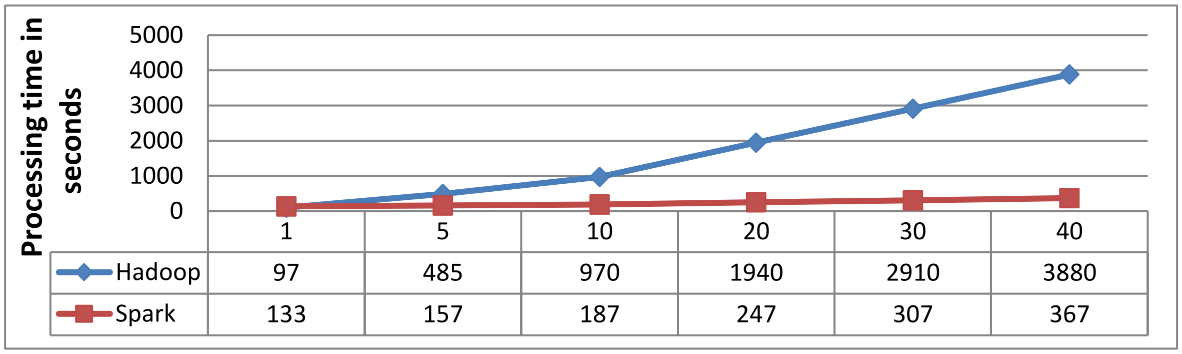
	500MB	1GB	2GB	5GB	10GB	20GB	40GB
🟦 Hadoop	102	234	386	620	992	1452	1762
🟥 Spark	54	169	288	502	882	1323	1630

Fig. 8. Variation in data size over processing time in seconds on wikilinks dataset



	500MB	1GB	2GB	5GB	10GB	20GB	40GB
🟦 Hadoop	131	267	432	696	1025	1598	1889
🟥 Spark	66	184	307	582	965	1473	1750

Fig. 9. Variation in data size over processing time in seconds on wikilanguage dataset

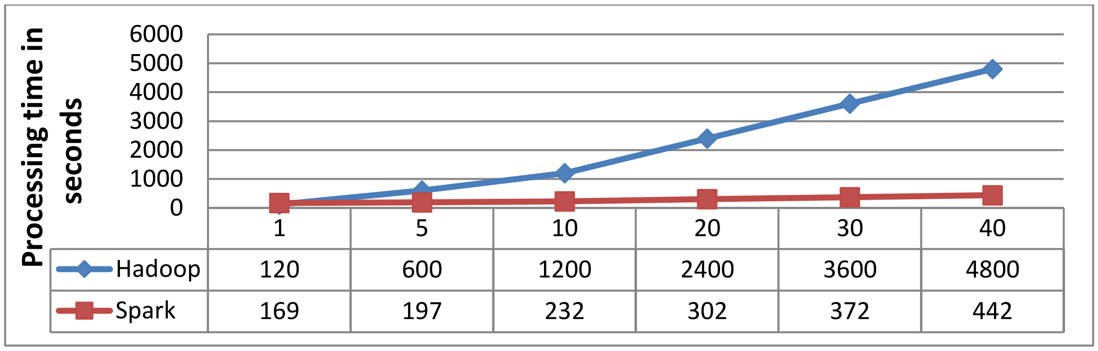


	1	5	10	20	30	40
Hadoop	97	485	970	1940	2910	3880
Spark	133	157	187	247	307	367

Fig. 10. Logistic regression performance in Hadoop and Spark on wikilinks dataset

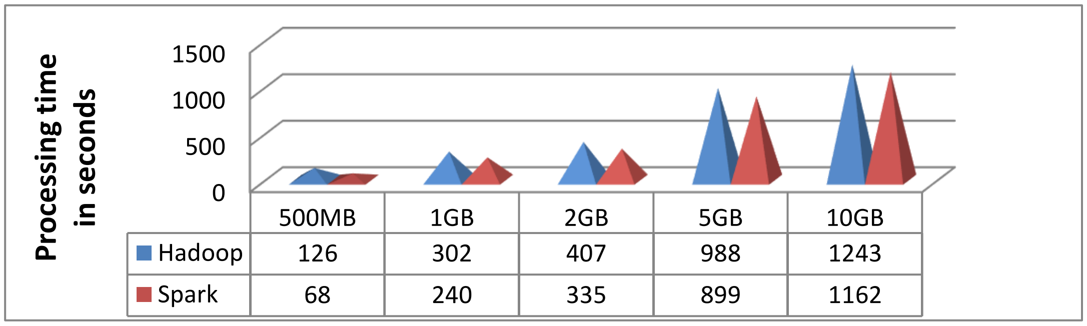
5.1 Analysis Using Word Count

For this experimental evaluation, two benchmark datasets i.e. wiki-links and wiki-language have been taken in various combinations of sizes (500MB- 10GB) and iterations. The processing time of these various groupings of data sizes over Hadoop MapReduce and Spark is recorded. A word count algorithm with the varying data sizes has been taken for evaluation. Based on the results obtained, a comparative analysis is presented here.



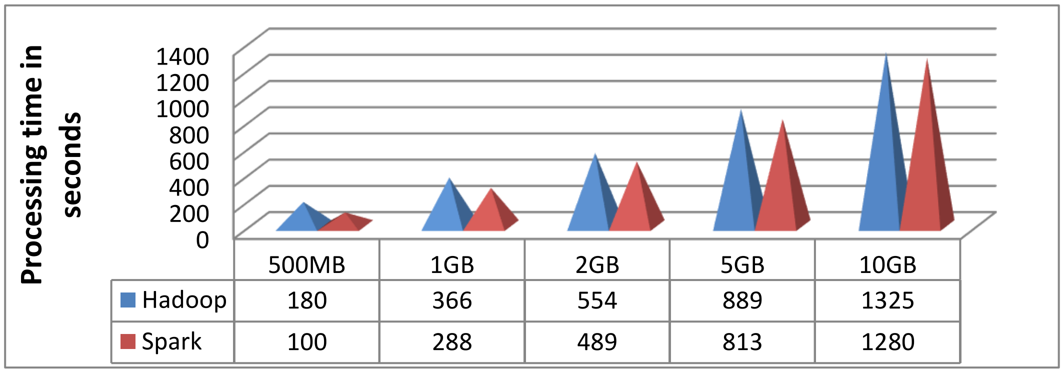
	1	5	10	20	30	40
Hadoop	120	600	1200	2400	3600	4800
Spark	169	197	232	302	372	442

Fig. 11. Logistic regression performance in Hadoop and Spark on wikilanguage dataset



	500MB	1GB	2GB	5GB	10GB
🟦 Hadoop	126	302	407	988	1243
🟥 Spark	68	240	335	899	1162

Fig. 12. Data size variation over processing time on wikilinks dataset



	500MB	1GB	2GB	5GB	10GB
🟦 Hadoop	180	366	554	889	1325
🟥 Spark	100	288	489	813	1280

Fig. 13. Variation in data size over processing time on wiki-language dataset

The suitability and performance of these distributed frameworks are also shown with the help of graphs.

The figure 8 and 9 show the performance of Hadoop MapReduce and Spark based on the variation of data sizes over time using word count algorithm.

Two benchmark datasets, wikilinks, and wikilanguages have been considered for experimental evaluation and various combination of data sizes such as 500MB, 1GB, 2.5GB, 5GB, 10GB, 20GB, and 40GB have been used to see the performance of Hadoop and Spark frameworks with regard to the processing time.

After plotting a graph, it is clear that Spark is better than the Hadoop MapReduce due to its In-memory processing capability that is absent in Hadoop MapReduce disk-based processing.

5.2 Analysis Using Logistic Regression

For further analysis based on logistic regression algorithm the same benchmark datasets, i.e. wikilinks and wiki-language have been taken. The analysis is purely based on iterations rather than various combinations of data sizes. Time taken on each level of iterations over Hadoop MapReduce and Spark is recorded.

A logistic regression with varying iterations based on processing time has been taken for evaluation. Based on the results obtained, a comparative analysis is presented here. The performance and suitability of these distributed model-based frameworks are also shown through the help of graphs.

The figures 10 and 11 shows processing time-based performance evaluation of the logistic regression algorithm at various iterations. In the shown graph, the x-axis indicates various iterations and the y-axis indicates the processing time taken by the algorithm in seconds.

For performing the logistic regression, a dataset, which has the size of 40GB, has been used. The two-benchmark dataset wiki-links and wiki-language are used here. All the computation is performed on Spark and Hadoop frameworks. On various iteration sizes, which are 1, 5, 10, 20, 30, and 40 times have been used to evaluate performance based on processing time over the existing distributed frameworks.

On wiki-link dataset, Hadoop takes 97 sec in every iterative cycle and Spark takes 133 sec in the first iteration but for later iterations, it takes only 6 sec. Due to the feature of In-memory computation, the Spark reuses the cache data in future iterations, which enhance the overall performance as, indicated in figure 10.

On wiki-language dataset, Hadoop takes 120 sec in every iterative cycle and Spark takes 169 sec in the first iteration but for later iterations, it takes only 7 sec more as shown in figure 11.

5.3 Analysis Using Distributed K-Means Clustering Algorithm

The experimental evaluation of distributed K-Means clustering is performed using Hadoop MapReduce and Spark frameworks. Four benchmark datasets i.e. Wiki language, Wikilinks, Enron and Wikipedia dataset are used so far. Various combinations of data sizes and increased number of iterations are used for result findings based on their processing time. To obtain more generalized results the performance and behavior of both frameworks are exhaustively observed.

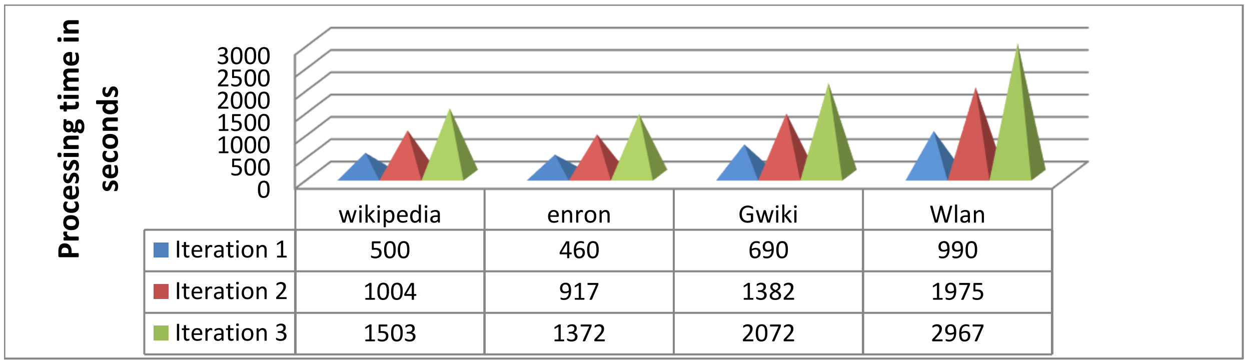
Wiki-links dataset with different combinations of sizes (500MB to 10GB) has been shown in figure 12. The processing time (in seconds) of various data sizes i.e. 500MB, 1GB, 2GB, 5GB, and 10GB is analyzed by using both distributed frameworks, which is shown in the graph After analyzing the graph, it is clearly visible that performance (based on processing time elapse) of Spark is better than the Hadoop MapReduce.

The wiki-language dataset with different combinations of sizes (500MB to 10GB) has been shown in figure 13. The processing time (in seconds) of various data sizes i.e. 500MB, 1GB, 2GB, 5GB, and 10GB is analyzed by using both distributed frameworks, which is shown in the graph.

After analyzing the graph, it is clearly visible that performance (based on processing time elapse) of Spark is better than the Hadoop MapReduce.

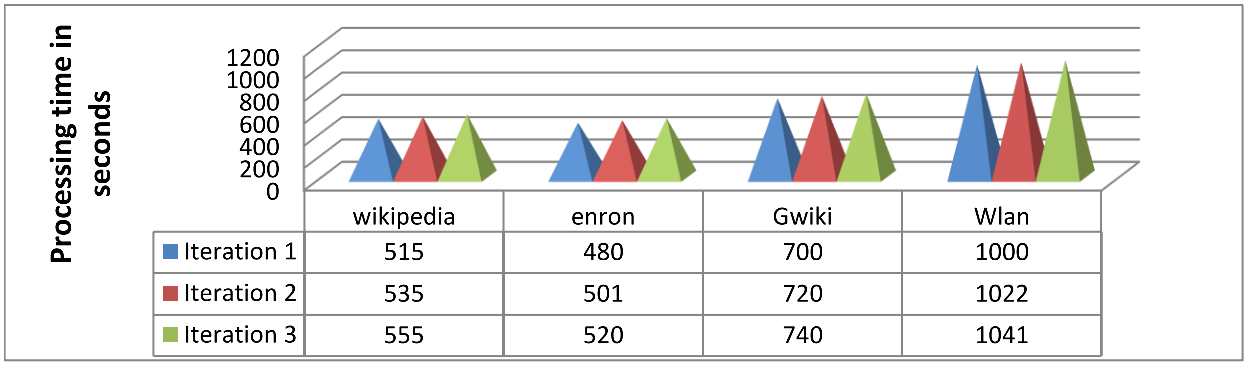
Four benchmark datasets having the size of 5GB each is taken i.e. Wikipedia, Enron, Wikilinks and wiki language and its analysis is shown in figure 14. Varying iterations and processing time in seconds are taken as constraints in x-axis and y-axis respectively on the graph.

All the computation is performed on the Hadoop, which is an On-disk computation model. We calculate the processing time elapses in seconds at each iteration. In iteration one the Hadoop takes, 460 seconds, 500 seconds, 990 seconds and 690 seconds., In the second iteration it takes 917 seconds, 1004 seconds, 1975 seconds and, 1382 seconds and in the third iteration it takes 1372 seconds, 1503 seconds, 2967 seconds and 2072 seconds to process Enron, Wikipedia, Wikilanguage and Wikilinks and dataset respectively.



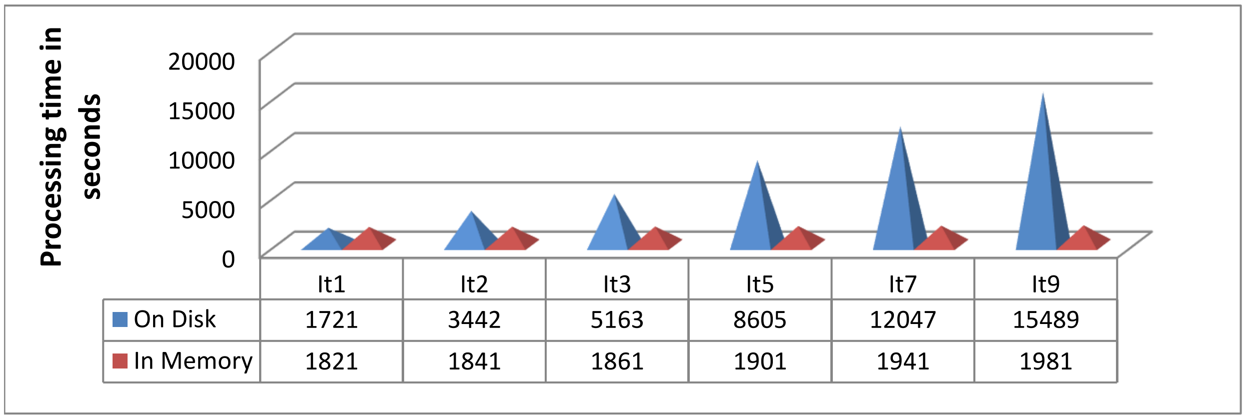
	wikipedia	enron	Gwiki	Wlan
🟦 Iteration 1	500	460	690	990
🟥 Iteration 2	1004	917	1382	1975
🟩 Iteration 3	1503	1372	2072	2967

Fig. 14. Performance analysis based on the iteration of Distributed K-Means in Hadoop



	wikipedia	enron	Gwiki	Wlan
🟦 Iteration 1	515	480	700	1000
🟥 Iteration 2	535	501	720	1022
🟩 Iteration 3	555	520	740	1041

Fig. 15. Performance analysis based on the iteration of Distributed K-Means in Spark



	It1	It2	It3	It5	It7	It9
🟦 On Disk	1721	3442	5163	8605	12047	15489
🟥 In Memory	1821	1841	1861	1901	1941	1981

Fig. 16. Performance analysis based on the iteration of distributed k-means in Hadoop and Spark

Having a size of 5GB the all four-benchmark dataset is taken i.e. Wikipedia, Enron, Wikilinks and Wikilanguage and its analysis are shown in figure 15. Varying iterations and processing time in seconds are taken as constraints in x-axis and y-axis respectively on the graph.

All the computation is performed on the Hadoop, which is an On-disk computation model.

We calculate processing time elapses in seconds at each iteration. In iteration one Spark takes 480 seconds, 515 seconds, 1000 seconds and 700 seconds to process Enron, Wikipedia, Wikilanguage and Wikilinks dataset respectively but in later iterations, Spark takes only 20 more seconds in subsequent iterations. The cached data is reused by the Spark so that in later iterations its performance is enhanced.

After analyzing figure 14 and 15, it is observed that the performance and speed (based on pro-cessing time elapse) of Spark is more the Hadoop because Spark follows In-memory computation mod-el which is far better than Hadoop's On-disk computation model.

The benchmark wiki-language dataset having 40GB of size has been taken for analysis using a distributed clustering algorithm (distributed K-means). The distributed K-means clustering algorithm is developed for both distributed frameworks (Hadoop and Spark) and its analysis is shown in figure 16.

Varying iterations and processing time in seconds are taken as constraints in x-axis and y-axis respectively on the graph.

All the computation is performed on the Hadoop and Spark, which are an On-disk, based computation model and In-memory based computation model respectively.

We calculate processing time elapses in seconds at various iteration such as 9 time, 7 times, 5 times, 3 times and 1 time. Spark and Hadoop take 1721 and 1821 seconds in the first iteration respectively. In subsequent iterations, Hadoop takes same as it takes in the first iteration i.e. 1721 seconds but Spark takes only 20 more seconds in subsequent iterations.

Due to the feature of In-memory computation, the Spark reuses the cache data in future iterations, which enhance the overall performance as, indicated in the graph.

6 Conclusion

In this research work, a comparative analysis of Hadoop and Spark has been presented based on working principle, performance, cost, ease of use, compatibility, data processing, failure tolerance, and security. Experimental analysis has also been performed to observe the performance of Hadoop and Spark for establishing their suitability under different constraints of the distributed computing environment.

From the experimental analysis, it is clearly observed that the Hadoop, which is an On-disk computation-based model, is lacking behind in terms of performance than In-memory based computation model that is Spark.

It is also visibly proven from the experimental results that the On-disk based computation is ten times slower than In-memory based computation. So, the In-memory based computation model (Spark) wins the battle which is two times faster than On-memory based computation model (Hadoop) and it is also observed that the algorithmic performance does not depend only on the size of dataset moreover it depends more on the richness of the corpus in the given dataset. Hadoop MapReduce framework is a milestone in Big Data analytics and it has given a backbone to technocrats for the development of a new programming framework.

As a period of time, it is also noticed that the MapReduce is not capable enough to resolve all the needs of the distributed environment but still researchers prefer it for research, experimentation and data manipulation.

On the other hand, the Spark is up-to-date and having many additional feature especially In-memory data processing. Although both frameworks are equally compatible as per recent trends, Hadoop MapReduce is better in terms of cost, security and fault tolerance and most suitable for platforms for batch processing while Spark is more suitable for real-time data processing. In the future, these frameworks and some new one like shark may be tested more rigorously on an experimental basis for establishing their impact and suitability.

References

1. **Jacobs, A. (2009).** The pathologies of big data. *Communications of the ACM*, Vol. 52, No. 8, pp. 36– 44.
2. **Zikopoulos, P. & Eaton, C. (2011).** *Understanding big data: Analytics for enterprise class Hadoop and streaming data.* McGraw-Hill Osborne Media.

3. **Kaisler, S., Armour, F., Espinosa, J.A., & Money, W. (2013).** Big data: Issues and challenges moving forward. *46th Hawaii International Conference on System Sciences,* pp. 995–1004. DOI: 10.1109/HICSS.2013.645.
4. **Spark (2013).** https://Spark.apache.org/Intro to Spark: stanford.edu/~rezab/Sparkclass/ slides/itas_workshop.pdf
5. **Liu, X., Han, J., Zhong, Y., Han, C., & He, X. (2009).** August. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS. *IEEE International Conference on Cluster Computing and Workshops*, pp. 1–8. DOI: 10.1109/CLUSTR.2009.5289196.
6. **YARN (2013).** http://hadoop.apache.org/
7. **Jiang, L., Li, B., & Song, M. (2010).** The optimization of HDFS based on small files. *3ed IEEE international conference on broadband network and multimedia technology (IC-BNMT),* pp. 912–915. DOI: 10.1109/ICBNMT.2010.5705223.
8. **Mackey, G., Sehrish, S., & Wang, J., (2009).** Improving metadata management for small files in HDFS. *IEEE international conference on cluster computing and workshops*, pp. 1–4. DOI: 10.1109/CLUSTR.2009.5289133.
9. **Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., & Qin, X. (2010).** Improving mapreduce performance through data placement in heterogeneous Hadoop clusters. *Processing, Workshops and Phd Forum (IPDPSW),* pp. 1–9. DOI: 10.1109/IPDPSW.2010.5470880.
10. **Thanh, T.D., Mohan, S., Choi, E., Kim, S., & Kim, P. (2008).** A taxonomy and survey on distributed file systems. *Fourth International Conference on Networked Computing and Advanced Information Management*, Vol. 1, pp. 144–149. DOI: 10.1109/NCM.2008.162.
11. **Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., & Stoica, I. (2010).** Spark: Cluster computing with working sets. *HotCloud,* Vol. 10, pp. 1–7.
12. **Cito, J., Leitner, P., Fritz, T., & Gall, H.C. (2015).** The making of cloud applications: An empirical study on software development for the cloud. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering,* pp. 393–403). DOI: 10.1145/2786805.2786826.
13. **Cavallaro, G., Riedel, M., Benediktsson, J.A., Goetz, M., Runarsson, T., Jonasson, K., & Lippert, T. (2014).** Smart data analytics methods for remote sensing applications. *IEEE geoscience and remote sensing symposium*, pp. 1405–1408. DOI: 10.1109/IGARSS.2014.6946698.
14. **Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., & Stoica, I. (2012).** Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation,* pp. 2.
15. **Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., & Saha, B. (2013).** Apache Hadoop Yarn: Yet another resource negotiator. *Proceedings of the 4th Annual Symposium on Cloud Computing,* No. 5, pp. 1–16. DOI: 10.1145/2523616.2523633.
16. **Ketu, S., Prasad, B.R., & Agarwal, S. (2015).** Effect of corpus size selection on performance of map-reduce based distributed k-means for big textual data clustering. *Proceedings of the Sixth International Conference on Computer and Communication Technology,* pp. 256–260. DOI: 10.1145/2818567.2818653.
17. **Nandimath, J., Banerjee, E., Patil, A., Kakade, P., Vaidya, S., & Chaturvedi, D. (2013).** Big data analysis using Apache Hadoop. *IEEE 14th International Conference on Information Reuse & Integration (IRI),* pp. 700–703. DOI: 10.1109/IRI.2013.6642536.
18. **Gu, L. & Li, H. (2013).** Memory or time: Performance evaluation for iterative operation on Hadoop and spark. *IEEE 10th International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing,* pp. 721–727. DOI: 10.1109/HPCC.and.EUC.2013.106.
19. **Chen, M., Mao, S., & Liu, Y. (2014).** Big data: A survey. *Mobile networks and applications,* Vol. 19, No. 2, pp. 171–209. DOI: 10.1007/s11036-013-0489-0.
20. **Shinnar, A., Cunningham, D., Saraswat, V., & Herta, B. (2012).** M3R: increased performance for in-memory Hadoop jobs. *Proceedings of the VLDB Endowment,* Vol. 5, No. 12, pp. 1736–1747.
21. **Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., & Murthy, R. (2009).** Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, Vol. 2, No.2, pp.1626–1629.
22. **Xin, R.S., Rosen, J., Zaharia, M., Franklin, M.J., Shenker, S., & Stoica, I. (2013).** Shark: SQL and rich analytics at scale. *Proceedings of the ACM SIGMOD International Conference on Management of data*, pp. 13–24. DOI: 10.1145/ 2463676. 2465288.

23. **Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013).** Discretized streams: Fault-tolerant streaming computation at scale. *Proceedings of the twenty-fourth ACM symposium on operating systems principles*, pp. 423–438. DOI: 10.1145/2517349.2522737.
24. **Marz, N., Xu, J., & Jackson, J. (2013).** *Storm.*
25. **Garg, N. (2013).** *Apache Kafka.* Packt Publishing Ltd.
26. **Owen, S., Anil, R., Dunning, T., & Friedman, E. (2011).** *Mahout in action*. Manning Publications.
27. **WIKIPEDIA (2013).** http://wiki.dbpedia.org/
28. **CMU (2012).** https://www.cs.cmu.edu/~./enron/
29. **DBPEDIA.ORG (2014).** http://wiki.dbpedia.org/

Article received on30/10/2019; accepted 05/03/2020. Corresponding author is Shwet Ketu.