A Comparative Study of Big Data Frameworks

Article *in* International Journal of Computer Science and Information Security, · January 2019

CITATIONS
16

READS
4,932

3 authors:

Safaa Alkatheri
King Abdulaziz University
7 PUBLICATIONS **29** CITATIONS
SEE PROFILE

Samah Abbas
King Abdulaziz University
8 PUBLICATIONS **37** CITATIONS
SEE PROFILE

Muazzam Ahmed Siddiqui
King Abdulaziz University
41 PUBLICATIONS **648** CITATIONS
SEE PROFILE

A Comparative Study of Big Data Frameworks

Safaa Alkatheri[1]
Department of Information Systems
Faculty of Computing and Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia
Safaa-alkatheri@hotmail.com

Samah Anwar Abbas[2]
Department of Management Information Systems
Faculty of Economics and Administration
King Abdulaziz University, Jeddah, Saudi Arabia
sabbas@kau.edu.sa

Muazzam Ahmed Siddiqui[3]
Department of Information Systems
Faculty of Computing and Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia
maasiddiqui@kau.edu.sa

***Abstract*—Every day, big data analytics is gaining more popularity as a tool for analyzing significant amounts of data on demand. Four of the most common big data processing frameworks include Apache Hadoop, Apache Spark, Apache Storm, and Apache Flink. While all four support big data processing, these frameworks differ in their usage and the underlying architecture that supports this usage. A number of studies have devoted time and effort to compare these big data frameworks by evaluating them for a defined key performance indicator (KPI). This paper summarizes these earlier efforts by identifying a common set of key performance indicators, which are Processing Time, CPU Consumption, Latency, Throughput, Execution Time, Sustainable Input Rate, Task Performance, Scalability, and Fault Tolerance, and comparing the four big data frameworks along these KPIs through a literature review. Our work identified Spark as a winner across multiple KPIs, which are processing time, CPU consumption, Latency, Execution time, task performance, and Scalability, for non real-time data, when compared with Apache Hadoop and Apache Storm frameworks. while Flink was best for stream processing in Processing time, CPU consumption, Latency, Throughput, Execution time, task performance, Scalability, and Fault tolerance, when compared with Apache Spark and Apache Storm frameworks.**

Keywords—Big data, performance evaluation, Flink, Hadoop, Spark, Storm

I. INTRODUCTION

In recent years, big data has become a more significant issue, which generates quickly due to rapid technological development. If anyone looks at the size of data, they will see that it is growing rapidly from Petabytes to Exabytes or Zetta_ bytes. Particularly, in social media analytics by using structured and unstructured data that collecting from several of social media channel. For example, on the Tumblr blog platform within seventy-two hours of being created, there were approximately thirty thousand new posts. On Twitter more than one hundred thousand Tweets, and posting on Facebook more than two hundred thousand pictures [1]. All of these numbers above indicate the volumes of data are exploding, and that data is growing faster than ever before. Due to that, the term “big data" has appeared and captured the attention of the computing field and researchers today.

The term “big data” was first introduced in 2005 by Roger Magoulas [2]. It is defined by him as a huge amount of data that cannot be managed or processed by traditional data management techniques. This data originates from various resources such as; social media sites, videos, sensors, smartphones, and search queries, to name a few. There are several important aspects that define “big data" from other data, which is: the huge size, the data sets that are composed of complex and independent data. Furthermore, it cannot be processed with traditional data management techniques [3].

As a result of huge data and the complexity that can happen when data is used for analyzing purposes, big data analytical tools are becoming one of the most important technologies. These technologies provide the ability to organize or manipulate all data, rather than using traditional platform of databases or systems.

The purpose of this review paper is to present an overview of four big data frameworks and compare them across a set of

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

predefined key performance indicators through literature review.

This paper has been divided as the following; the next part explains the main characteristics of big data, which is referred to as V's of big data. This is followed by the discussion of some big data frameworks, which are; Hadoop, Spark, Storm, and Flink. The categories of them will be presented as a comparative study between the frameworks and the obtained results. After that, we give some concluding points.

II. The Characteristics of Big Data

As we defined the meaning of Big Data in the previous stage, it is now important to illustrate its characteristics. It is composed of generic Big Data requirements (volume, variety, and velocity), which are collectively known as 3Vs [4]. Recently, the characteristics of Big Data evolved from 3V's to 6V's, adding the features of value, veracity, and variability. The latter three are referred to as acquired Big Data requirements after entering the system. Figure 1 shows the 6V's of Big Data.

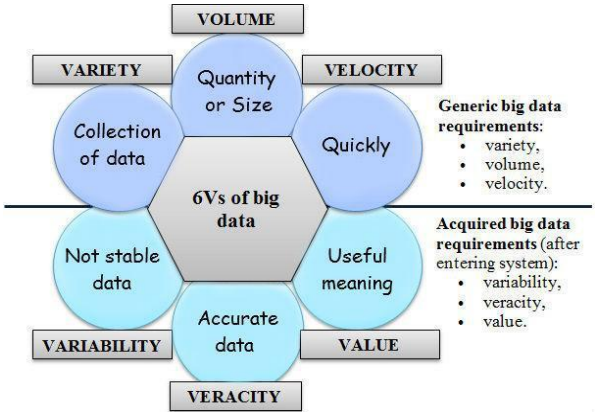

Fig. 1. 6V's of Big Data

A. Volume

Volume refers to the quantity or size of the data. The size of Big Data is in the order of Terabytes (TB), Petabytes (PB), Zettabytes (ZB), and Exabytes (EB) [7] [8]. Organizations such as Facebook, YouTube, Google, and NASA possess enormous amounts of data bringing new challenges to store, retrieve, analyze, and process this data. The use of Big Data rather than traditional storage has changed how we transfer data and use it [9].

B. Variety

Variety refers to the different types of data that is being generated. Variety can be measured using different dimensions such as structure enabling us to difference structured, semi-structured and unstructured data, or processing volume as in batch versus stream.

C. Variability

Variability refers to the data that is not stable, which cannot be easily dealt with, and difficult to manage. Explaining variable data amounts to a significant problem for researchers [6].

D. Velocity

Velocity refers to how quickly Big Data is generated in order to manipulate, exchange, store, and analyze [10]. Velocity presents new research challenges for data scientists because of the high costs involved [11]. When the user needs to retrieve or manipulate the data and the process is not adequately fast, the data is left behind [11].

E. Veracity

Veracity refers to the quality of data being processed. The veracity of the data source also depends on analyzing the data accuracy [5].

F. Value

Value refers to the purpose or the business outcome that the data brings in, to facilitate the decision-making process [5].

III. Big Data Processing Frameworks

The four frameworks compared in this paper differ from each other in terms of the features they support and their underlying architecture, while keeping the primary purpose of supporting big data processing at their core. This section presents an overview of the architectures of these four big data processing frameworks.

A. Hadoop

In 2008, Apache Hadoop was defined by Doug Cutting and Mike Cafarella as an open source framework, which collects and processes the distributed data through a group of host machines (hardware layer) called clusters or nodes. It provides a distribution services machine rather than one service. So, they can work in parallel by using clusters or nodes [12] [13].

Figure 2 illustrates the three main layers of Hadoop framework. The first one is the data storage layer for collecting data, which contains Hadoop Distributed File System (HDFS). The second layer is the YARN infrastructure, which provides arithmetic resources for job scheduling such as CPU and memory. The third is MapReduce, which is used for processing data (software layer) with other processes [13].

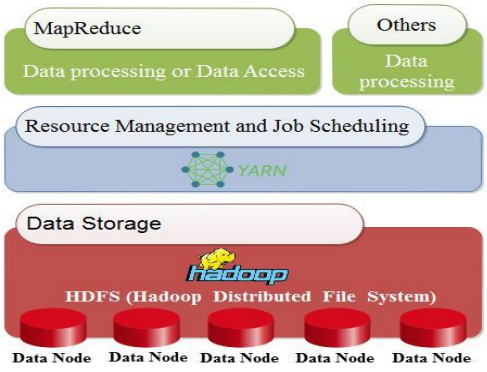

Fig. 2. Hadoop Architecture Adapted [5]

Numerous companies, enterprise, and organizations utilize

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

Apache Hadoop for two main reasons. First, conducting research for academic or scientific purposes. Second, engaging in the analysis to satisfy customers' needs and help organizations take the right decisions. For example, when the organization needs to know what kind of product customers require. Then, it can produce the product that is needed in abundance, which is one of the several applications of Apache Hadoop [12].

B. Spark

Apache Spark is an open source framework that was established at the University of California, Berkeley. It became an Apache project in 2013, providing faster services with large-scale data processing [14]. Spark framework is to Hadoop what MapReduce is to data processing and HDFS. In addition, Spark has data sharing known as Resilient Distributed Datasets (RDD) and Directed Acyclic Graph (DAG) [14].Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.

Figure 3 represents Spark architecture, which is very easy and fast for selecting a huge amount of data processing. Spark mainly consists of five layers. The first layer comprises of data storage systems such as HDFS and HBASE. The second layer is resource management; for instance, YARN and Mesos. The third is a Spark core engine. The fourth is a library, which is composed of SQL, stream processing, MLlib for machine learning, Spark R, and GraphX for graph processing [14]. The last layer is an application program interface such as Java or Scala. In general, Spark offers a large-scale data processing framework used by banks, telecommunication companies, game companies, governments, and firms such as Apple, Yahoo and Facebook.

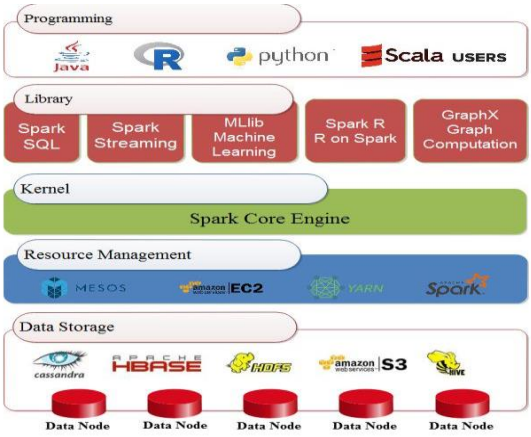

Fig. 3. Spark Architecture Adapted [15]

C. Storm

Storm engine [16] is an open source framework that was designed for processing streaming data in real-time. It is written in Clojure language [17]. Figure 4 shows that a storm process can work with any program language and on any application development platforms. So, it guarantees that data will not be lost.

Figure 5 illustrates the two types of nodes: The first is the master node and the second is the worker node. The master node is used for monitoring failures, taking the responsibility of distributed node, and specifying each task for each machine. All these tasks are collectively known as Nimbus, which is similar to Hadoop's Job-Tracker [18]. The worker node is called Supervisor. It works when Nimbus assigns a specific process to it. Thus, each sub-process of a topology works with many distributed machines. Zookeeper plays the role of coordinator between Nimbus and the Supervisors. More importantly, if there is a failure in any cluster, it reassigns the task to another one. So, the slave node controls the execution of its own tasks.

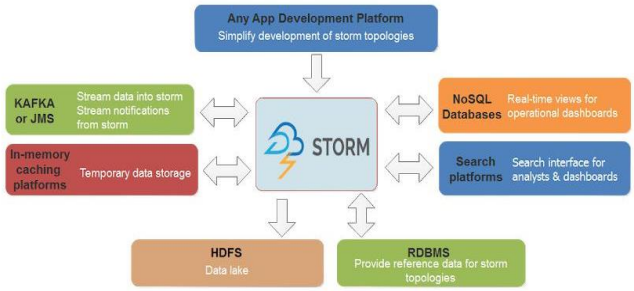

Fig. 4. Storm Architecture [19]

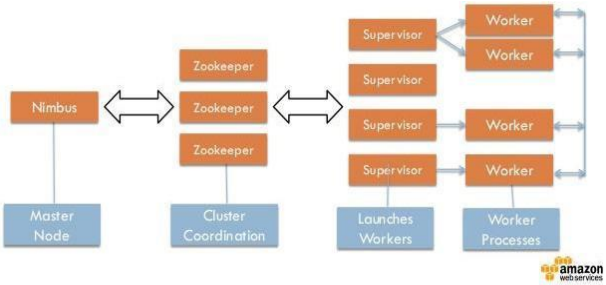

Fig. 5. Storm Processing [20]

D. Flink

Apache Flink [21] is an open source framework that was created in 2010 by three German universities, and it has been used effectively for processing data both in realtime and batch mode. It uses in-memory processing technique and provides a number of APIs such as stream processing API (data stream), batch processing API (data set), and table API that has been used for queries. It has machine learning (ML) and graph processing (Gelly) libraries as well.

Figure 6 illustrate the architecture of Flink [22]. In the base layer, the storage layer can read and write the data from multiple destinations such as HDFS, local files, and so on. Then, the deployment and resource management layer contains the cluster manager for managing the tasks of planning, monitoring the jobs, and managing the resources. This layer also contains the environment that executes the program, which are the clusters or cloud environments. Besides, it has the local area for single Java virtual machine.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

Moreover, it has the kernel layer for distributed stream data flow engine for real-time processing. Also, the application program has interface layers for two processes: batch and streaming. The upper layer is a library in which the program is written in Java or Scala programming language. It is then submitted to the compiler for conversion with the help of the Flink optimizer in order to improve its performance.

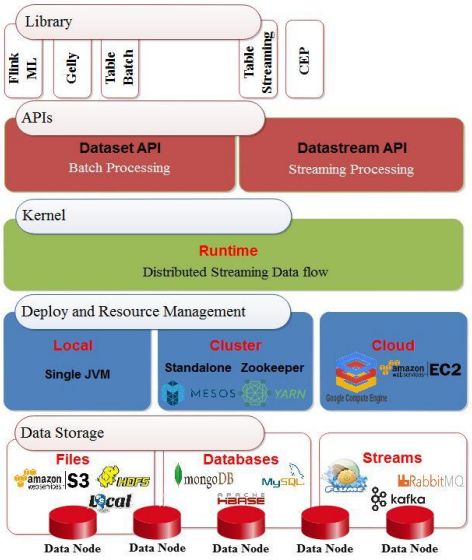

Fig. 6. Flink Architecture Adapted [23]

IV. Feature Comparison of Big Data Frameworks

Each of the big data frameworks in our study supports a set of features, which could be used as a key performance indicator too. In this section, we will present a set of common features identified through literature review and compare the four frameworks across these features.

A. Scalability

Scalability is the ability of a system to respond to increasing amount of load. It has two types: scale up (vertically) and scale out (horizontally). Scale up is used to upgrade the hardware configuration, whereas scale out is used to add extra hardware. All the four frameworks in our study are horizontally scalable. It means we can add many nodes to the cluster as and when required.

B. Message Delivery Guarantees

Message delivery guarantees are used in the case of failure. According to the four frameworks mentioned above, it can be divided into two types: exactly once delivery and at-least-once delivery. Exactly once delivery means that the message will not be duplicated, nor be lost, and will deliver to the recipient exactly once. On the other hand, at-least-once delivery means there are many attempts to deliver the message and at least one of these attempts succeeds. In addition, the message can be duplicated without being lost.

C. Computation Mode

Computation mode could be in-memory computing or the more "traditional" mode where computation results are written back to the disk. In-memory computing is faster but comes at a potential disadvantage of losing the contents in case of the machine being turned off.

D. Auto-Scaling

Auto-scaling refers to the automatic scaling of cloud services, either up or down, based on the situation.

E. Iterative Computation

Iterative computation refers to the implementation of an iterative method that estimates an approximate solution in the absence of a real solution or when the cost of a real solution is prohibitively high.

TABLE I. Summarization of Some Features of Big Data Frameworks

Features	***Hadoop***	***Spark***	***Storm***	***Flink***
Processing Mode	Batch	Batch and Stream	Stream	Batch and Stream
Scalability	Horizontal	Horizontal	Horizontal	Horizontal
Message Delivery Guarantees	Exactly once	Exactly once	At least once	Exactly once
Computation Mode	Disk-based	In memory	In memory	In memory
Auto-scaling	Yes	Yes	No	No
Iterative Computation	Yes	Yes	Yes	Yes

V. Literature Review Comparing the Four Big Data Processing Frameworks

This section presents existing literature comparing the aforementioned four big data processing frameworks. Through the literature, we identified nine different key performance indicators, namely, processing time, CPU consumption, latency, throughput, execution time, sustainable input rate, task performance, scalability, and fault tolerance.

A. Processing Time

A number of existing studies have assessed the performance of big data frameworks through processing time. One of the works that employed this measure as a key performance indicator was conducted by [24]. This study used a personalized monitoring tool in order to monitor resource usage, and Python script to detect the states of machines. In the batch mode experiment, the researchers included a dataset of 10 billion tweets, while in the stream mode experiment, they collected one billion tweets. In terms of batch mode, they evaluated the impact of data size and the used cluster on processing time. Regarding the size of data, they found that Spark was faster than Hadoop and Flink, and that Flink was the slowest. They also note that Flink was faster than Hadoop only when datasets were small (less than five GB). In fact, compared to Spark, which avoids input/output operations, Hadoop transferred data by accessing the HDFS; thus, in this

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

case, processing time was affected by the amount of input/output operations and as such, processing time increased when processing large amounts of data. On the other hand, regarding the size of the used cluster, the study demonstrated that Hadoop and Flink take a longer time than Spark, as the execution of jobs in Spark was influenced by the number of processors and the amount of read/write operations on RAM, rather than disk use, as in the case of Hadoop. In the stream mode experiment, the researchers studied processing rate by evaluating the impact of window time on the number of processed events. They demonstrated that Flink and Storm had the best processing rates, better than Spark, in the case of sending a tweet of 100 KB per message; this was because these frameworks used different values for window time. Flink and Storm use milliseconds, while Spark uses seconds. On the other hand, Flink worked more efficiently than Storm and Spark in the case of sending five tweets of 500 KB per message. Additionally, in a study conducted by [25], the authors evaluated the performance of both Flink and Spark, based on E-commerce data from the Amazon website. The dataset they used was in the JSON format. In addition, each record had a fixed number of fields and the average size of a record was 3000 bytes. They found the average time for processing data by using Flink to be 240.3sec, while this decreased for Spark to 60.4sec. Therefore, the performance of Spark was better than that of Flink, by approximately 179.5%.

B. CPU Consumption

A number of authors have used CPU consumption for assessing performance of big data frameworks. In a study conducted by [24], Flink was found to use fewer resources than Hadoop and Spark in the case of batch mode. This is because Flink partially exploits disk and memory resources, compared to Spark and Hadoop. Additionally, based on stream mode, the study found that Flink was lower than Spark and Storm in terms of CPU consumption, because Flink is primarily designed to process large messages, compared to Storm. Spark collects events every second and then performs the task; as such, more than one message is processed and as a result, high CPU usage is incurred. In a study conducted by [26], the authors used the Yahoo streaming benchmark (YSB) and three data streaming frameworks -Spark, Storm, and Flink- to conduct their experiment. They found Storm to have the highest CPU resource usage, compared to the other frameworks. Additionally, a study conducted by [27] found that Apache Spark reaches approximately 100% CPU utilization, while Apache Flink executed the same load using less CPU resources.

C. Latency

Latency is another important performance measures for assessing the performance of big data frameworks. For example, [28] used the RAM3S framework to compare the performance of Spark, Storm, and Flink, using a dataset from surveillance cameras that included 3425 videos of 1595 different people. The researchers implemented their experiment in a local environment, as well as on the Google Cloud platform. When the number of nodes for local clusters and the cloud varied, they found that Apache Storm achieved the lowest latency, and was very similar to Flink latency. However, Spark attained the highest latency, due to its micro-batch design. Furthermore, a study conducted by [26] found that Spark could outperform Flink only if a high latency was acceptable. In addition, the authors of [29] used the RAM3S framework to compare the real time analysis of significantly large multimedia flows in Storm, Spark, and Flink. They used the YouTube Faces Dataset (YTFD), which included 3425 videos of 1595 different people, and different video resolutions, where 480 360 is the most common, and a total of 621, 126 frames, which connected with the minimum face on average for 181.3 frames per video. They demonstrated that Storm and Flink achieved slightly better results than Spark. Furthermore, a study conducted by [30] compared Spark and Storm based on two groups of datasets, i.e. 3000 benign and 500 anomalies. The first dataset was from the cluster of Spark in VMware (D1), and the second from the Yahoo Cloud Serving Benchmark (YCSB), predicting anomaly (D2). The authors tested the data in different VMs and in a single VM in order to complete their experiments. They found that the average latency in Spark was less than in Storm in all cases.

D. Throughput

Throughput is another measure that has been used for assessing the performance of big data frameworks. For example, [28] found that Spark attained lower throughput than Storm and Flink, while in [26], the researchers demonstrated that when the batching interval was longer in Spark, the throughput was higher. In addition, the study conducted by [29] show that Storm and Flink achieved slightly better results than Spark in the case of using the cloud environment, without considering the time needed for building the D-stream.

E. Execution Time

Execution time was used by [31] to evaluate and compare the performance of Hadoop, Spark, and Flink frameworks. They performed their experiment on DAS-4 using the Big Data Evaluator tool (BDEv), in order to automate the configuration of frameworks. They note that excluding TeraSort, as well as putting Spark and Flink in the place of Hadoop, lead to reduce the time of execution by 77% and 70% on average, respectively, when 49 nodes were used. In work conducted by [32], the researchers evaluated the performance of Spark and Hadoop in terms of WordCount and logistic regression program, using an open source dataset that included a prediction of bankruptcy for various companies. Their results demonstrated that the time of execution for the WordCount program in Spark was less than for Hadoop. In addition, the time for executing the logistic regression program in Spark was less than for Hadoop. For example, if the number of iterations was 100, the time of execution in Spark was 3.452sec; for Hadoop, it was 9.383sec. Therefore, Spark outperformed Hadoop in both WordCount and logistic regression. One of the reasons for this is that using the cache in the memory storage of Spark made the process faster. Moreover, in a study conducted by [33], the authors measured performance based on the WordCount program using Spark and the MapReduce framework, which runs on single node Hadoop (HDFS), installed on an Ubuntu machine. They used a dataset in the

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

form of a large text le, which comprised customer reviews and feedback for multiple products, and distributed this file into different sizes. They found that Spark was able to perform faster, roughly three-tofour times, compared to the MapReduce programming framework. In addition, the study conducted by [27] compared Spark and Flink frameworks using Karamel (web application) in order to evaluate performance at system level and application level. The data generated using the TeraSort application and stored using HDFS, as well as various input levels (200GB, 400GB, and 600GB) were used. The researchers found that Flink decreased execution time, which was 1.5 times faster than Spark for Terasort application.

F. Sustainable Input Rate

A study conducted by [28], used sustainable input rate as a performance measure to compare big data frameworks. The measure was used when the number of computing nodes for the local cluster and cloud varied. They demonstrated that Storm outperformed Flink and Spark in both scenarios (local and cloud). This result was due to the simplest at-least-once semantics employed by Storm, while in Flink, this is exactlyonce semantics. In addition, the topology of Storm is defined by the programmer, while in Flink, it is defined by the optimizer. This led to reduced efficiency in Flink. On the other hand, Spark was not mainly designed to be a streaming engine; thus, the management of streaming was one of the reasons for inferior input rates.

G. Task Performance

Another study conducted by [31] compared the performance of big data frameworks on a number of given tasks including WordCount, k-means, PageRank, Grep, TeraSort, and connected components. The study found that Spark achieved the best in WordCount and k-means, compared to Flink and Hadoop, while Flink achieved better results for PageRank. On the other hand, both Flink and Spark achieved the same results for Grep, TeraSort, and connected components, and outperformed Hadoop in these measures. One of the interpretations that led to the result of WordCount was that Spark uses a reduceByKey() function in order to sum the number of times each word appears, compared to Flink, which uses a groupBy().sum() function, which is less optimized. As a result, Flink suffers from fewer memory optimizations. In Grep, Spark and Flink performed better than Hadoop, because Hadoop uses one MapReduce to search the pattern and another to sort the results; this led to a high number of memory copies and writes to HDFS. In PageRank, Flink achieved the best performance, because it uses delta iterations that process only elements that have not yet reached their final value.

H. Scalability

In terms of measuring scalability, the authors in [34] connected the plan of the operator's execution (end-to-end execution time) with the resource use and parameter configurations in order to measure the performance of Spark and Flink. They showed that Spark was roughly 1.7x faster than Flink, particularly in big graph processing. Contrastingly, with a large dataset and fixed node, Flink was better, outperforming Spark by 10%.

I. Fault Tolerance

In terms of fault tolerance measure, the study conducted by [29] points out that Flink has higher fault tolerance than both the Storm and Spark frameworks. Overall, all of the studies reviewed here indicate that Spark is the best in terms of measuring processing time, compared to Hadoop and Flink. Also in terms of latency, it was better if VMs and single VM was used to detect anomalies. In addition, it was the best in terms of throughput as well as execution time when compared to Hadoop and Flink. Also in term of WordCount and k-means, it was better compared to Flink and Hadoop. Moreover, it was also better compared to Hadoop in term of Grep, TeraSort, and Connected Components. Additionally, in term of scalability, it was better in the case of big graph processing compared to Flink.

Flink was more efficient in the measuring of processing time, compared to Storm and Spark. In addition, it was more efficient in throughput in the case of using the cloud environment, without considering the time for building the d-stream. In addition to that, it was better in execution time compared to Spark only in the case of using the Karamel and TeraSort applications. Moreover, in term of PageRank, it was the best compared to Spark and Hadoop. Also, it was better than Hadoop in term of Grep, TeraSort, and connected components. In term of scalability, it was the best compared to Spark only if the dataset is large and the number of nodes is fixed. Again, it was better than Storm and Spark in term of fault tolerance.

Storm had the best performance in the measure of CPU utilization compared to Spark, Flink, and Hadoop frameworks. In addition, it had the best latency compared to Spark and Flink. Also, it had the best throughput only in the case of using the cloud environment, without considering the time needed for building the d-stream. Moreover, it had a better sustainable input rate compared to Flink and Spark. Table 2 shows summarization of the literature of comparing the four big data frameworks.

TABLE II. SUMMARIZATION OF THE LITERATURE OF COMPARING THE FOUR BIG DATA FRAMEWORKS

Categorized	***In case of***	***Hadoop***	***Spark***	***Flink***	***Storm***
Processing time [24]	Big data set	Less faster	Fastest	Slower	Not Compar-ed
Processing time [24]	Small data set	Slower	Fastest	Less faster	Not Compar-ed
Processing time [24]	Cluster size	Slow	Fast	Slow	Not Compar-ed
Processing time [24]	Sending a tweet of 100 KB per message	Not Compar-ed	Slow	Fast	Fast
Processing time [24]	Sending five tweets of 500 KB per message.	Not Compar-ed	Slow	Fast	Slow
Processing time [25]	JSON format	Not Compar	Fast	Slow	Not Compar-

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

Categorized	*In case of*	*Hadoop*	*Spark*	*Flink*	*Storm*
	data set	-ed			ed
CPU consumpt-ion [24]	Batch mode	High CPU usage	High CPU usage	Less CPU usage	Not Compar-ed
CPU consumpt-ion [24]	Stream mode	Not Compar-ed	High CPU usage	Less CPU usage	High CPU usage
CPU consumpt-ion [26]	Stream mode	Not Compar-ed	Less higher CPU usage	Less CPU usage	Highest CPU usage
CPU consumpt-ion [27]	Batch mode	Not Compar-ed	High CPU usage	Less CPU usage	Not Compar-ed
Latency [28]	RAM3S frame-work	Not Compar-ed	High latency	Low latency	Low latency
Latency [29]	RAM3S frame-work	Not Compar-ed	High latency	Low latency	Low latency
Latency [30]	Using different group of data set	Not Compar-ed	Less latency	Not Compar-ed	High latency
Throughput [28]	RAM3S frame-work	Not Compar-ed	Low throughp ut	High throughp ut	High throughp ut
Throughput [29]	Using cloud environ-ment	Not Compar-ed	Low throughp ut	High throughp ut	High throughp ut
Execution time [31]	DAS-4 and Tera Sort	High executio n time	Low execution time	Low execution time	Not Compar-ed
Execution time [32]	Word Count and logistic regress-ion program	High executio n time	Low execution time	Not Compar-ed	Not Compar-ed
Execution time [33]	Word Count	High executio n time	Low execution time	Not Compar-ed	Not Compar-ed
Execution time [27]	Tera Sort	Not Compar-ed	High execution time	Low execution time	Not Compar-ed
Sustainable input-rate [28]	Different local and cloud cluster	Not Compar-ed	Low sustainabl e input rate	Low sustainabl e input rate	High sustainab le input rate
Word Count, k-means [31]	Word Count, kmeans	Worse	Best	Worse	Not Compar-ed
PageRank [31]	PageRank	Worse	Worse	Best	Not Compar-ed
Grep,Tera Sort,and connected components [31]	Grep, Tera,Sort, and connect-ed compo-nents	Worse	Best	Best	Not Compar-ed
Scalability [34]	Big graph process-ing	Not Compar-ed	Best	Worse	Not Compar-ed
Scalability [34]	Large dataset and fixed	Not Compar-ed	Worse	Best	Not Compar-ed
	node				
Fault tolerance [29]	Fault tolerance	Not Compar-ed	Low	High	Low

VI. Conclusion and Future Work

In this paper, we analyzed and compared four frameworks, Hadoop, Flink, Spark, and Storm, based on different key performance indicators for measuring their performance. The results of this study show that Flink performed the best compared to the other frameworks, as it achieved the best performance in eight measures. Spark was better than the other frameworks in six measures, and Storm was better than the other frameworks in four measures. Thus, users from companies, researchers, as well as individuals who are interested in this field can choose the appropriate framework, based on the key performance indicators they wish to use, in order to analyze data and gain efficient results. At the end, they will gain high performance in computing (HPC). In future, by considering these measurements in the performance of the four frameworks, the opportunity for enhancement is possible for each framework in any measure that has low impact in terms of gaining HPC. As such, we aspire to seeing enhancements in some of these frameworks, while also including other frameworks that are able of delivering high performance.

References

[1] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *Int. J. Inf. Manag.*, vol. 35, no. 2, pp. 137–144, Apr. 2015.

[2] M. R. Wigan and R. Clarke, "Big data's big unintended consequences," *Computer*, vol. 46, no. 6, pp. 46–53, 2013.

[3] "Open Framework, Information Management Strategy & Collaborative Governance | Data & Social Methodology - MIKE2.0 Methodology." [Online]. Available: http://mike2.openmethodology.org/. [Accessed: 23-Jul-2018].

[4] D. Laney, "3D data management: Controlling data volume, velocity and variety," *META Group Res. Note*, vol. 6, no. 70, p. 1, 2001.

[5] X. Jin, B. W. Wah, X. Cheng, and Y. Wang, "Significance and Challenges of Big Data Research," *Big Data Res.*, vol. 2, no. 2, pp. 59–64, Jun. 2015.

[6] W. Fan and A. Bifet, "Mining big data: current status, and forecast to the future," *ACM SIGKDD Explor. Newsl.*, vol. 14, no. 2, pp. 1–5, 2013.

[7] P. Zikopoulos and C. Eaton, *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[8] M. Barkhordari and M. Niamanesh, "ScaDiPaSi: an effective scalable and distributable MapReduceBased method to find patient similarity on huge healthcare networks," *Big Data Res.*, vol. 2, no. 1, pp. 19–27, 2015.

[9] "9 Trends to Watch in the Growing Big Data Market," *Database Trends and Applications*, 16-Feb2016. [Online]. Available: http://www.dbta.com/BigDataQuarterly/Articles/9-Trends-to-Watch-in-theGrowing-Big-Data-Market-109143.aspx. [Accessed: 23-Jul-2018].

[10] "7 Important Big Data Trends for 2016," *Datafloq*. [Online]. Available: https://datafloq.com/read/7- big-data-trends-for-2016/1699. [Accessed: 23-Jul-2018].

[11] H. G. Miller and P. Mork, "From data to decisions: a value chain for big data," *It Prof.*, vol. 15, no. 1, pp. 57–59, 2013.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

[12] "Welcome to ApacheTM Hadoop®!" [Online]. Available: http://hadoop.apache.org/. [Accessed: 23-Jul2018].

[13] R. A. Fadnavis and S. Tabhane, "Big Data Processing Using Hadoop," *Int. J. Comput. Sci. Inf. Technol.*, vol. 6, no. 1, pp. 443–445, 2015.

[14] "Apache SparkTM - Unified Analytics Engine for Big Data." [Online]. Available: https://spark.apache.org/. [Accessed: 23-Jul-2018].

[15] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on Apache Spark," *Int. J. Data Sci. Anal.*, vol. 1, no. 3–4, pp. 145–164, 2016.

[16] A. Toshniwal et al., "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 147–156.

[17] "Apache Storm." [Online]. Available: http://storm.apache.org/. [Accessed: 23-Jul-2018].

[18] "Apache Storm: Architecture - DZone Big Data," *dzone.com*. [Online]. Available: https://dzone.com/articles/apache-storm-architecture. [Accessed: 23-Jul-2018].

[19] "Apache Storm," *Hortonworks*. [Online]. Available: https://hortonworks.com/apache/storm/. [Accessed: 23-Jul-2018].

[20] F. Follow *et al.*, "Amazon Web Services." [Online]. Available: https://www.slideshare.net/AmazonWebServices. [Accessed: 23-Jul-2018].

[21] A. Alexandrov et al., "The stratosphere platform for big data analytics," *VLDB Journal— Int. J. Very Large Data Bases*, vol. 23, no. 6, pp. 939–964, 2014.

[22] "Apache Flink: Stateful Computations over Data Streams." [Online]. Available: https://flink.apache.org/. [Accessed: 23-Jul-2018].

[23] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.*, vol. 36, no. 4, 2015.

[24] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. M. Nguifo, "An experimental survey on big data frameworks," *Future Gener. Comput. Syst.*, 2018.

[25] D. Kaur, R. Chadha, and N. Verma, "INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY COMPARISON OF MICRO-BATCH AND STREARMING ENGINE ON REAL TIME DATA."

[26] Z. Karakaya, A. Yazici, and M. Alayyoub, "A Comparison of Stream Processing Frameworks," in *Computer and Applications (ICCA), 2017 International Conference on*, 2017, pp. 1–12.

[27] S. Perera, A. Perera, and K. Hakimzadeh, "Reproducible experiments for comparing apache flink and apache spark on public clouds," *ArXiv Prepr. ArXiv*161004493, 2016.

[28] I. Bartolini and M. Patella, "Comparing Performances of Big Data Stream Processing Platforms with RAM3S," in *Proc. SEBD*, 2017.

[29] I. Bartolini and M. Patella, "A general framework for real-time analysis of massive multimedia streams," *Multimed. Syst.*, pp. 1–16, 2017.

[30] M. Solaimani, M. Iftekhar, L. Khan, B. Thuraisingham, and J. B. Ingram, "Spark-based anomaly detection over multi-source VMware performance data in real-time," in *Computational Intelligence in Cyber Security (CICS), 2014 IEEE Symposium on*, 2014, pp. 1–8.

[31] J. Veiga, R. R. Expósito, X. C. Pardo, G. L. Taboada, and J. Tourifio, "Performance evaluation of big data frameworks for large-scale data analytics," in *Big Data (Big Data), 2016 IEEE International Conference on*, 2016, pp. 424–431.

[32] A. V. Hazarika, G. J. S. R. Ram, and E. Jain, "Performance comparision of Hadoop and spark engine," in *I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2017 International Conference on*, 2017, pp. 671–674.

[33] A. Verma, A. H. Mansuri, and N. Jain, "Big data management processing with Hadoop MapReduce and spark technology: A comparison," in *Colossal Data Analysis and Networking (CDAN), Symposium on*, 2016, pp. 1–4.

[34] O.-C. Marcu, A. Costan, G. Antoniu, and M. S. Pérez-Hernández, "Spark versus flink: Understanding performance in big data analytics frameworks," in *Cluster Computing (CLUSTER), 2016 IEEE International Conference on*, 2016, pp. 433–442.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500