Hindawi
Scientific Programming
Volume 2018, Article ID 5418679, 17 pages
https://doi.org/10.1155/2018/5418679

Review Article

Big Data in Cloud Computing: A Resource Management Perspective

Saeed Ullah, M. Daud Awan, and M. Sikander Hayat Khiyal

Faculty of Computer Science, Preston University, Islamabad, Pakistan

Correspondence should be addressed to Saeed Ullah; saeedullah@gmail.com

Received 27 October 2017; Revised 11 February 2018; Accepted 25 March 2018; Published 9 May 2018

Academic Editor: Sungyong Park

The modern day advancement is increasingly digitizing our lives which has led to a rapid growth of data. Such multidimensional datasets are precious due to the potential of unearthing new knowledge and developing decision-making insights from them. Analyzing this huge amount of data from multiple sources can help organizations to plan for the future and anticipate changing market trends and customer requirements. While the Hadoop framework is a popular platform for processing larger datasets, there are a number of other computing infrastructures, available to use in various application domains. The primary focus of the study is how to classify major big data resource management systems in the context of cloud computing environment. We identify some key features which characterize big data frameworks as well as their associated challenges and issues. We use various evaluation metrics from different aspects to identify usage scenarios of these platforms. The study came up with some interesting findings which are in contradiction with the available literature on the Internet.

1. Introduction

We live in the information age, and an important measurement of present times is the amount of data that is generated anywhere around us. Data is becoming increasingly valuable. Enterprises are aiming at unlocking data's hidden potential and deliver competitive advantage [1]. Stratistics MRC projected that the data analytics and Hadoop market, which accounted for $8.48 billion in 2015, is expected to reach at $99.31 billion by 2022 [2]. The global big data market has estimated that it will jump from $14.87 billion in 2013 to $46.34 billion in 2018 [3]. Gartner has predicted that data will grow by 800 percent over the next five years and 80 percent of the data will be unstructured (e-mails, documents, audio, video, and social media content) and 20 percent will be structured (e-commerce transactions and contact information) [1].

Today's largest scientific institution, CERN, produces over 200 PB of data per year in the Large Hadron Collider project (as of 2017). The amount of generated data on the Internet has already exceeded 2.5 exabytes per day. Within one minute, 400 hours of videos are uploaded on YouTube, 3.6 million Google searches are conducted worldwide each minute of every day, more than 656 million tweets are shared on Twitter, and more than 6.5 million pictures are shared on Instagram each day. When a dataset becomes so large that its storage and processing become challenging due to the constraints of existing tools and resources, the dataset is referred to as big data [4, 5]. It is the first part of the journey towards delivering decision-making insights. But instead of focusing on people, this process utilizes a much more powerful and evolving technology, given the latest breakthroughs in this field, to quickly analyze huge streams of data, from a variety of sources, and to produce one single stream of useful knowledge [6].

Big data applications might be viewed as the advancement of parallel computing, but with the important exception of the scale. The scale is the necessity arising from the nature of the target issues: data dimensions largely exceed conventional storage units, the level of parallelism needed to perform computation within a strict deadline is high, and obtaining final results requires the aggregation of large numbers of partial results. The scale factor, in this case, does not only have the same effect that it has in classical parallel computing, but

it surges towards a dimension in which automated resource management and its exploitation are of significant value [7].

An important factor for the success in big data analytical projects is the management of resources: these platforms use a substantial amount of virtualized hardware resources to optimize the tradeoff between costs and results. Managing such resources is definitely a challenge. Complexity is rooted in their architecture: the first level of complexity stems from their performance requirements of computing nodes: typical big data applications utilize massively parallel computing resources, storage subsystems, and networking infrastructure because of the fact that results are required within a certain time frame, or they can lose their value over time. Heterogeneity is a technological need: evolvability, extensibility, and maintainability of the hardware layer imply that the system will be partially integrated, replaced, or extended by means of new parts, according to the availability on the market and the evolution of technology [7]. Another important consideration of modern applications is the massive amount of data that need to be processed. Such data usually originate from different sets of devices (e.g., public web, business applications, satellites, or sensors) and procedures (e.g., case studies, observational studies, or simulations). Therefore, it is imperative to develop computational architectures with even better performance to support current and future application needs. Historically, this need for computational resources was provided by high-performance computing (HPC) environments such as computer clusters, supercomputers, and grids. In traditional owner-centric HPC environments, internal resources are handled by a single administrative domain [19]. Cluster computing is the leading architecture for this environment. In distributed HPC environments, such as grid computing, virtual organizations manage the provisioning of resources, both internal and external, to meet application needs [20]. However, the paradigm shift towards cloud computing has been widely discussed in more recent researches [19, 21], targeting the execution of HPC workloads on cloud computing environments. Although organizations usually prefer to store their most sensitive data internally (on-premises), huge volumes of big data (owned by the enterprises or generated by third parties) may be stored externally; some of it may already be on a cloud. Retaining all data sources behind the firewall may result in a significant waste of resources. Analyzing the data where it resides either internally or in a public cloud data center makes more sense [1, 22].

Even if cloud computing has to be an enabler to the growth of big data applications, common cloud computing solutions are rather different from big data applications. Typically, cloud computing solutions offer fine-grained, loosely coupled applications, run to serve large numbers of users that operate independently, from multiple locations, possibly on own, private, nonshared data, with a significant amount of interactions, rather than being mainly batch-oriented, and generally fit to be relocated with highly dynamic resource needs. Despite such differences, cloud computing and big data architectures share a number of common requirements, such as automated (or autonomic) fine-grained resource management and scaling related issues [7].

As cloud computing begins to mature, a large number of enterprises are building efficient and agile cloud environments, and cloud providers continue to expand service offerings [1]. Microsoft's cloud Hadoop offering includes Azure Marketplace, which runs Cloudera Enterprise, MapR, and Hortonworks Data Platform (HDP) in a virtual machine, and Azure Data Lake, which includes Azure HDInsight, Data Lake Analytics, and Data Lake Store as managed services. The platform offers rich productivity suites for database, data warehouse, cloud, spreadsheet, collaboration, business intelligence, OLAP, and development tools, delivering a growing Hadoop stack to Microsoft community. Amazon Web Services reigns among the leaders of cloud computing and big data solutions. Amazon EMR is available across 14 regions worldwide. AWS offers versions of Hadoop, Spark, Tez, and Presto that can work off data stored in Amazon S3 and Amazon Glacier. Cloud Dataproc is Google's managed Hadoop and Spark cluster to use fully managed cloud services such as Google BigQuery and Bigtable. IBM differentiates BigInsights with end-to-end advanced analytics. IBM BigInsights runs on top of IBM's SoftLayer cloud infrastructure and can be deployed on more than 30 global data centers. IBM is making significant investments in Spark, BigQuality, BigIntegrate, and IBM InfoSphere Big Match that run natively with YARN to handle the toughest Hadoop use cases [23].

In this paper, we give an overview of some of the most popular and widely used big data frameworks, in the context of cloud computing environment, which are designed to cope with the above-mentioned resource management and scaling problems. The primary object of the study is how to classify different big data resource management systems. We use various evaluation metrics for popular big data frameworks from different aspects. We also identify some key features which characterize big data frameworks as well as their associated challenges and issues. We restricted our study selection criteria to empirical studies from existing literature with reported evidence on performance evaluation of big data resource management frameworks. To the best of our knowledge, thus far there has been no empirical based performance evaluation report on major resource management frameworks. We investigated the validity of existing research by performing a confirmatory study. For this purpose, the standard performance evaluation tests as well as custom load test cases were performed on a 10+1 nodes t2.2xlarge Amazon AWS cluster. For experimentation and benchmarking, we followed the same process as outlined in our earlier study [24].

The study came up with some interesting findings which are in contradiction with the available literature on the Internet. The novelty of the study includes the categorization of cloud-based big data resource management frameworks according to their key features, comparative evaluation of the popular big data frameworks, and the best practices related to the use of big data frameworks in the cloud.

The inclusion and exclusion criteria for relevant research studies are as follows:

(i) We selected only those resource management frameworks for which we found empirical evidence of being offered by various cloud providers.

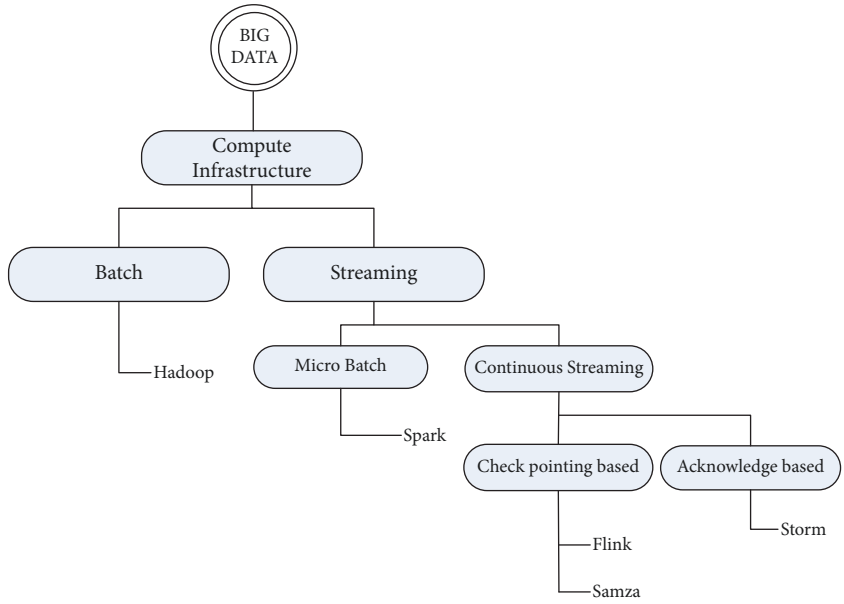

Figure 1: Classification of big data resource management frameworks.

(ii) Several vendors offer their proprietary solutions for big data analysis which could be the potential candidate for comparative analysis being conducted in this study. However, these frameworks were not selected based on two reasons. Firstly, most of these solutions are the extension of open-source solution and hence these exhibit the identical perform results in most of the cases. Secondly, for our empirical studies, researchers mostly prefer open-source solutions as the documentation, usage scenarios, source code, and other relevant details are freely available. Hence, we selected open-source solutions for the performance evaluation.

(iii) We did not include the frameworks which are now deprecated or discounted, such as Apache S4, in favor of other resource management systems.

This paper is organized as follows. Section 2 reviews the popular resource management frameworks. The comparison of big data frameworks is presented in Section 3. Based on the comparative evaluation, we categorize these systems in Section 4. Related work is presented in Section 5 and, finally, we present conclusion and possible future directions in Section 6.

2. Big Data Resource Management Frameworks

Big data is offering new emerging trends and opportunities to unearth operational insight towards data management. The most challenging issues for organizations are often that the amount of data is massive which needs to be processed at an optimal speed to synthesize relevant results. Analyzing such huge amount of data from multiple sources can help organizations plan for the future and anticipate changing market trends and customer requirements. In many of the cases, big data is analyzed in batch mode. However, in many situations, we may need to react to the current state of data or analyze the data that is in motion (data that is constantly coming in and needs to be processed immediately). These applications require a continuous stream of often unstructured data to be processed. Therefore, data is continuously analyzed and cached in memory before it is stored on secondary storage devices. Processing streams of data works by filtering in-memory tables of data across a cluster of servers. Any delay in the data analysis can seriously impact customer satisfaction or may result in project failure [25].

While the Hadoop framework is a popular platform for processing huge datasets in parallel batch mode using commodity computational resources, there are a number of other computing infrastructures that can be used in various application domains. The primary focus of this study is to investigate popular big data resource management frameworks which are commonly used in cloud computing environment. Most of the popular big data tools available for cloud computing platform, including the Hadoop ecosystem, are available under open-source licenses. One of the key appeals of Hadoop and other open-source solutions is the low total cost of ownership. While proprietary solutions have expensive license fees and may require more costly specialized hardware, these open-source solutions have no licensing fees and can run on industry-standard hardware [14]. Figure 1 demonstrates the classification of various styles of processing architectures of open-source big data resource management frameworks.

In the subsequent section, we discuss various open-source big data resource management frameworks that are

widely used in conjunction with cloud computing environment.

2.1. Hadoop. Hadoop [26] is a distributed programming and storage infrastructure based on the open-source implementation of the MapReduce model [27]. MapReduce is the first and current de facto programming environment for developing data-centric parallel applications for parsing and processing large datasets. The MapReduce is inspired by Map and Reduce primitives used in functional programming. In MapReduce programming, users only have to write the logic of Mapper and Reducer while the process of shuffling, partitioning, and sorting is automatically handled by the execution engine [14, 27, 28]. The data can either be saved in the Hadoop file system as unstructured data or in a database as structured data [14]. Hadoop Distributed File System (HDFS) is responsible for breaking large data files into smaller pieces known as blocks. The blocks are placed on different data nodes, and it is the job of the NameNode to notice what blocks on which data nodes make up the complete file. The NameNode also works as a traffic cop, handling all access to the files, including reads, writes, creates, deletes, and replication of data blocks on the data nodes. A pipeline is a link between multiple data nodes that exists to handle the transfer of data across the servers. A user application pushes a block to the first data node in the pipeline. The data node takes over and forwards the block to the next node in the pipeline; this continues until all the data, and all the data replicas, are saved to disk. Afterwards, the client repeats the process by writing the next block in the file [25].

The two major components of Hadoop MapReduce are job scheduling and tracking. The early versions of Hadoop supported limited job and task tracking system. In particular, the earlier scheduler could not manage non-MapReduce tasks and it was not capable of optimizing cluster utilization. So, a new capability was aimed at addressing these shortcomings which may offer more flexibility, scaling, efficiency, and performance. Because of these issues, Hadoop 2.0 was introduced. Alongside earlier HDFS, resource management, and MapReduce model, it introduced a new resource management layer called Yet Another Resource Negotiator (YARN) that takes care of better resource utilization [25].

YARN is the core Hadoop service to provide two major functionalities: global resource management (ResourceManager) and per-application management (ApplicationMaster). The ResourceManager is a master service which controls NodeManager in each of the nodes of a Hadoop cluster. It includes a scheduler, whose main task is to allocate system resources to specific running applications. All the required system information is tracked by a Resource Container which monitors CPU, storage, network, and other important resource attributes necessary for executing applications in the cluster. The ResourceManager has a slave NodeManager service to monitor application usage statistics. Each deployed application is handled by a corresponding ApplicationMaster service. If more resources are required to support the running application, the ApplicationMaster requests the NodeManager and the NodeManager negotiates with the ResourceManager (scheduler) for the additional capacity on behalf of the application [26].

2.2. Spark. Apache Spark [29], originally developed as Berkeley Spark, was proposed as an alternative to Hadoop. It can perform faster parallel computing operations by using in-memory primitives. A job can load data in either local memory or a cluster-wide shared memory and query it iteratively with much great speed as compared to disk-based systems such as Hadoop MapReduce [27]. Spark has been developed for two applications where keeping data in memory may significantly improve performance: iterative machine learning algorithms and interactive data mining. Spark is also intended to unify the current processing stack, where batch processing is performed using MapReduce, interactive queries are performed using HBase, and the processing of streams for real-time analytics is performed using other frameworks such Twitter's Storm. Spark offers programmers a functional programming paradigm with data-centric programming interfaces built on top of a new data model called Resilient Distributed Dataset (RDD) which is a collection of objects spread across a cluster stored in memory or disk [28]. Applications in Spark can load these RDDs into the memory of a cluster of nodes and let the Spark engine automatically manage the partitioning of the data and its locality during runtime. This versatile iterative model makes it possible to control the persistence and manage the partitioning of data. A stream of incoming data can be partitioned into a series of batches and is processed as a sequence of small-batch jobs. The Spark framework allows this seamless combination of streaming and batch processing in a unified system. To provide rapid application development, Spark provides clean, concise APIs in Scala, Java, and Python. Spark can be used interactively from the Scala and Python shells to rapidly query big datasets.

Spark is also the engine behind Shark, a complete Apache Hive-compatible data warehousing system that can run much faster than Hive. Spark also supports data access from Hadoop. Spark fits in seamlessly with the Hadoop 2.0 ecosystem (Figure 2) as an alternative to MapReduce, while using the same underlying infrastructure such as YARN and the HDFS. Spark is also an integral part of the SMACK stack to provide the most popular cloud-native PaaS such as IoT, predictive analytics, and real-time personalization for big data. In SMACK, Apache Mesos cluster manager (instead of YARN) is used for dynamic allocation of cluster resources, not only for running Hadoop applications but also for handling heterogeneous workloads.

The GraphX and MLlib libraries include state-of-the-art graph and machine learning algorithms that can be executed in real time. BlinkDB is a novel parallel, sampling-based approximate query engine for running interactive SQL queries that trade off query accuracy for response time, with results annotated by meaningful error bars. BlinkDB has been proven to run 200 times faster than Hive within an error rate of 2–10%. Moreover, Spark provides an interactive tool called Spark Shell which allows exploiting the Spark cluster in real time. Once interactive applications are created, they may subsequently be executed interactively in the cluster.

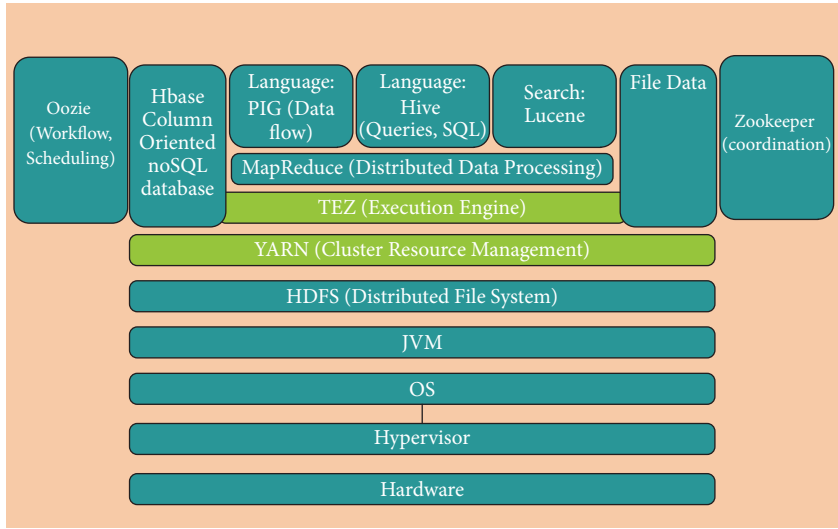

Figure 2: Hadoop 2.0 ecosystem, source: [14].

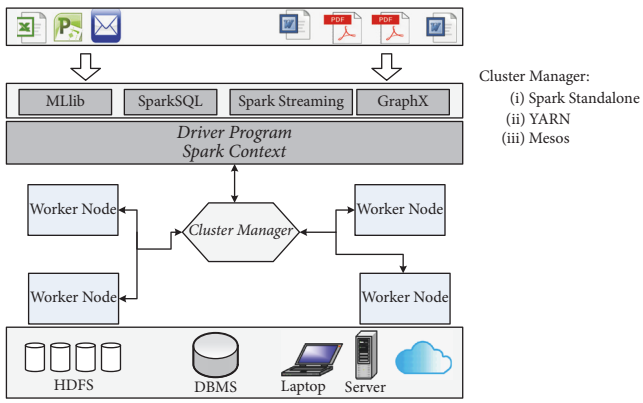

Figure 3: Spark architecture, source: [15].

In Figure 3, we present the general Spark system architecture.

2.3. Flink. Apache Flink is an emerging competitor of Spark which offers functional programming interfaces, much similar to Spark. It shares many programming primitives and transformations in the same way as what Spark does for iterative development, predictive analysis, and graph stream processing. Flink is developed to fill the gap left by Spark, which uses minibatch streaming processing instead of a pure streaming approach. Flink ensures high processing performance when dealing with complex big data structures such as graphs. Flink programs are regular applications which are written with a rich set of transformation operations (such as mapping, filtering, grouping, aggregating, and joining) to the input datasets. The Flink dataset uses a table-based model; therefore application developers can use index numbers to specify a particular field of a dataset [27, 28].

Flink is able to achieve high throughput and a low latency, thereby processing a bundle of data very quickly. Flink is designed to run on large-scale clusters with thousands of nodes, and in addition to a standalone cluster mode, Flink provides support for YARN. For distributed environment, Flink chains operator subtasks together into tasks. Each task is executed by one thread [16]. Flink runtime consists of two types of processes: there is at least one JobManager (also called masters) which coordinates the distributed execution. It schedules tasks, coordinates checkpoints, and coordinates recovery on failures. A high-availability setup may involve multiple JobManagers, one of which one is always the leader, and the others are standby. The TaskManagers (also called workers) execute the tasks (or, more specifically, the subtasks) of a dataflow/buffer and exchange the data streams. There must always be at least one TaskManager. The JobManagers and TaskManagers can be started in various ways: directly on the machines as a standalone cluster, in containers, or managed by resource frameworks like YARN or Mesos. TaskManagers connect to JobManagers, announcing themselves as available, and are assigned work. Figure 4 demonstrates the main components of Flink framework.

2.4. Storm. Storm [17] is a free open-source distributed stream processing computation framework. It takes several characteristics from the popular actor model and can be used with practically any kind of programming language for developing applications such as real-time streaming analytics, critical work flow systems, and data delivery services. The engine may process billions of tuples each day in a fault-tolerant way. It can be integrated with popular resource management frameworks such as YARN, Mesos, and Docker. Apache Storm cluster is made up of two types of processing actors: spouts and bolts.

(i) Spout is connected to the external data source of a stream and is continuously emitting or collecting new data for further processing.

(ii) Bolt is a processing logic unit within a streaming processing topology; each bolt is responsible for a

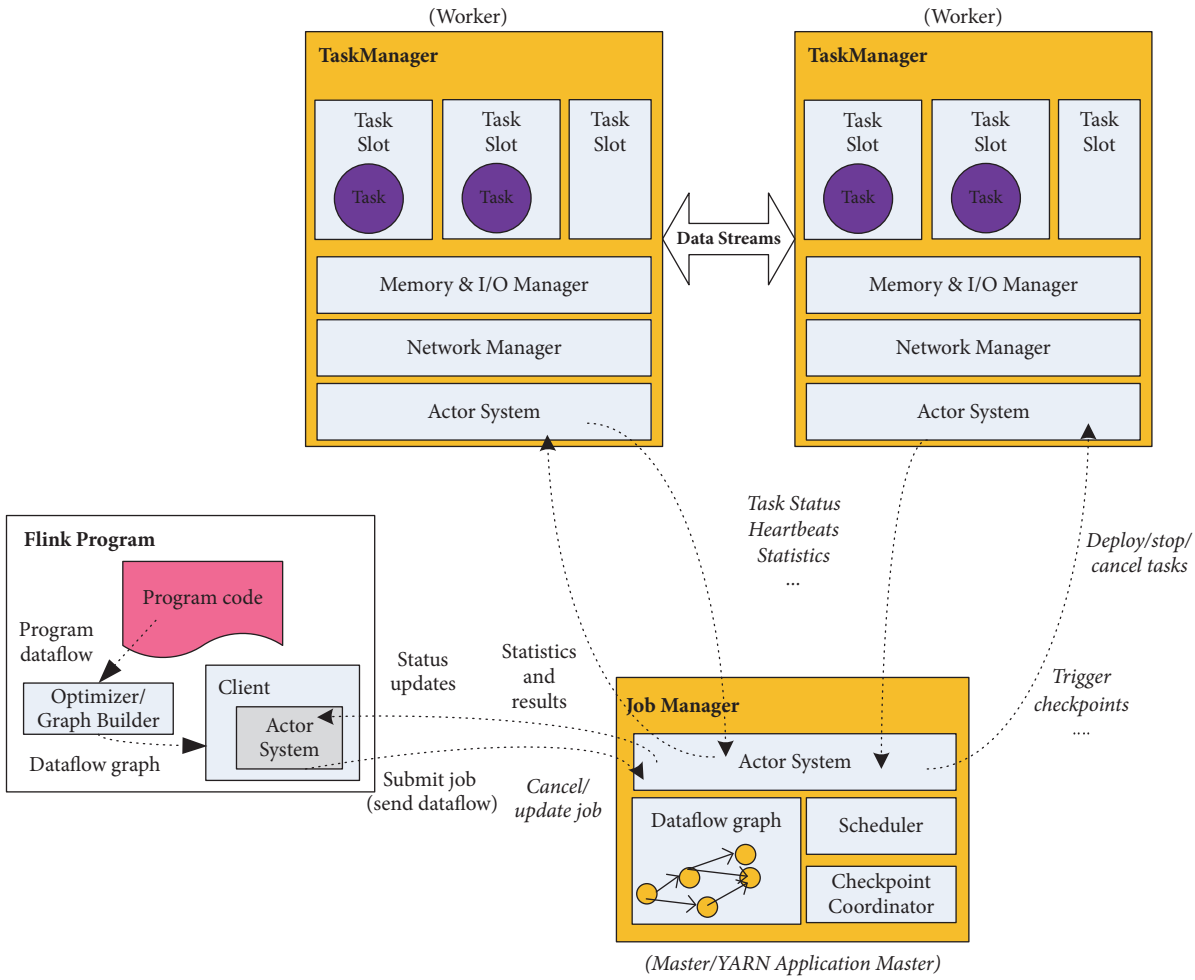

Figure 4: Architectural components of Flink, source: [16].

certain processing task such as transformation, filtering, aggregating, and partitioning.

Storm defines workflow as directed acyclic graphs (DAGs), called topologies with connected spouts and bolts as vertices. Edges in the graph define the link between the bolts and the data stream. Unlike batch jobs being only executed once, Storm jobs run forever until they are killed. There are two types of nodes in a Storm cluster: nimbus (master node) and supervisor (worker node). Nimbus, similar to Hadoop JobTracker, is the core component of Apache Storm and is responsible for distributing load across the cluster, queuing and assigning tasks to different processing units, and monitoring execution status. Each worker node executes a process known as the supervisor which may have one or more worker processes. Supervisor delegates the tasks to worker processes. Worker process then creates a subset of topology to run the task. Apache Storm does rely on an internal distributed messaging system, called Netty, for the communication between nimbus and supervisors. Zookeeper manages the communication between real-time job trackers (nimbus) and supervisors (Storm workers). Figure 5 outlines the high-level view of Storm cluster.

2.5. Apache Samza. Apache Samza [18] is a distributed stream processing framework, mainly written in Scala and Java. Overall, it has a relatively high throughput as well as somewhat increased latency when compared to Storm [8]. It uses Apache Kafka, which was originally developed for LinkedIn, for messaging and streaming, while Apache Hadoop YARN/Mesos is utilized as an execution platform for overall resource management. Samza relies on Kafka's semantics to define the way streams are handled. Its main objective is to collect and deliver massively large volumes of event data, in particular, log data with a low latency. A Kafka system's architecture is comparatively simple as it only consists of a set of brokers which are individual nodes that make up a Kafka cluster. Data streams are defined by topics, which is a stream of related information that consumers can subscribe to. Topics are divided into partitions that are distributed over the broker instances for retrieving the corresponding messages using a pull mechanism. The basic flow of job execution is presented in Figure 6.

Tables 1 and 2 present a brief comparative analysis of these frameworks based on some common attributes. As shown in the tables, MapReduce computation data flow follows chain of stages with no loop. At each stage, the program

Table 1: Comparison of big data frameworks.

Attribute	Framework				
	Hadoop	Spark	Storm	Samza	Flink
Current stable version	2.8.1	2.2.0	1.1.1	0.13.0	1.3.2
Batch processing	Yes	Yes	Yes	No	Yes
Computational model	MapReduce	Streaming (microbatches)	Streaming (microbatches)	Streaming	Supports continuous flow streaming, microbatch, and batch
Data flow	Chain of stages	Directed acyclic graph	Directed acyclic graphs (DAGs) with spouts and bolts	Streams (acyclic graph)	Controlled cyclic dependency graph through machine learning
Resource management	YARN	YARN/Mesos	HDFS (YARN)/Mesos	YARN/Mesos	Zookeeper/YARN/Mesos
Language support	All major languages	Java, Scala, Python, and R	Any programming language	JVM languages	Java, Scala, Python, and R
Job management/optimization	MapReduce approach	Catalyst extension	Storm-YARN/3rd-party tools like Ganglia	Internal JobRunner	Internal optimizer
Interactive mode	None (3rd-party tools like Impala can be integrated)	Interactive shell	None	Limited API of Kafka streams	Scala shell
Machine learning libraries	Apache Mahout/H2O	Spark ML and MLlib	Trident-ML/Apache SAMOA	Apache SAMOA	Flink-ML
Maximum reported nodes (scalability)	Yahoo Hadoop cluster with 42,000 nodes	8000	300	LinkedIn with around a hundred node clusters	Alibaba customized Flink cluster with thousands of nodes

Table 2: Comparative analysis of big data resource frameworks ($s = 5$).

	Hadoop	Spark	Flink	Storm	Samza
Processing speed	★★★	★★★★	★★★★★	★★★★	★★★★
Fault tolerance	★★★★	★★	★★★★	★★★	★★★★
Scalability	★★★★★	★★★★	★★★	★★★	★★★
Machine learning	★★	★★★★★	★★★★	★★★	★★★★
Low latency	★★	★★★	★★★	★★★★	★★★★
Security	★★★★	★★★★★	★★★★	★★★★	✘
Dataset size	★★★★★	★★★	★★★★	★★★	★★★

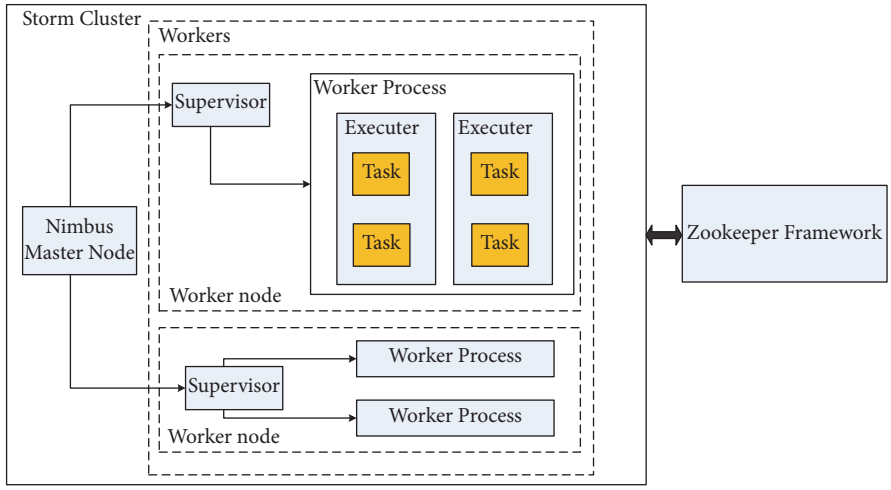

Figure 5: Architecture of Storm Cluster, source: [17].

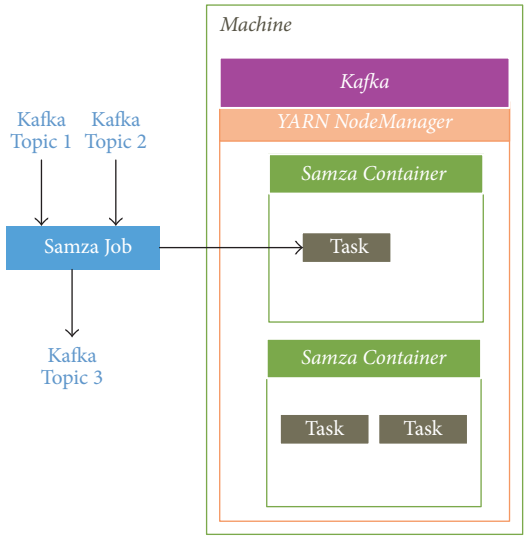

Figure 6: Samza architecture, source: [18].

proceeds with the output from the previous stage and generates an input for the next stage. Although machine learning algorithms are mostly designed in the form of cyclic data flow, Spark, Storm, and Samza represent it as directed acyclic graph to optimize the execution plan. Flink supports controlled cyclic dependency graph in runtime to represent the machine learning algorithms in a very efficient way. Hadoop and Storm do not provide any default interactive environment. Apache Spark has a command-line interactive shell to use the application features. Flink provides a Scala shell to configure standalone as well as cluster setup. Apache Hadoop is highly scalable and it has been used in the Yahoo production consisting of 42000 nodes in 20 YARN clusters. The largest known cluster size for Spark is of 8000 computing nodes while Storm has been tested on a maximum of 300 node clusters. Apache Samza cluster, with around a hundred nodes, has been used in LinkedIn data flow and application messaging system. Apache Flink has been customized for Alibaba search engine with a deployment capacity of thousands of processing nodes.

3. Comparative Evaluation of Big Data Frameworks

Big data in cloud computing, a popular research trend, is posing significant influence on current enterprises, IT industries, and research communities. There are a number of disruptive and transformative big data technologies and solutions that are rapidly emanating and evolving in order to provide data-driven insight and innovation. Furthermore, modern cloud computing services are offering all kinds of big data analytic tools, technologies, and computing infrastructures to speed up the data analysis process at an affordable cost. Although many distributed resource management frameworks are available nowadays, the main issue is how to

select a suitable big data framework. The selection of one big data platform over the others will come down to the specific application requirements and constraints that may involve several tradeoffs and application usage scenarios. However, we can identify some key factors that need to be fulfilled before deploying a big data application in the cloud. In this section, based on some empirical evidence from the available literature, we discuss the advantages and disadvantages of each resource management framework.

3.1. Processing Speed. Processing speed is an important performance measurement that may be used to evaluate the effectiveness of different resource management frameworks. It is a common metric for the maximum number of I/O operations to disk or memory or the data transfer rate between the computational units of the cluster over a specific amount of time. Based on the context of big data, the average processing speed represented as $\overline{m}$, calculated after n iterations run, is the maximum amount of memory/disk intensive operations that can be performed over a time interval t_i:

$$\overline{m} = \frac{\sum_{i=1}^{n} m_i}{\sum_{i=1}^{n} t_i}. \tag{1}$$

Veiga et al. [30] conducted a series of experiments on a multicore cluster setup to demonstrate performance results of Apache Hadoop, Spark, and Flink. Apache Spark and Flink resulted to be much efficient execution platforms over Hadoop while performing nonsort benchmarks. It was further noted that Spark showed better performance results for operations such as WordCount and K-Means (CPU-bound in nature) while Flink achieved better results in PageRank algorithm (memory bound in nature). Mavridis and Karatza [31] experimentally compared performance statistics of Apache Hadoop and Spark on Okeanos IaaS cloud platform. For each set of experiments, necessary statistics related to execution time, working nodes, and the dataset size were recorded. Spark performance was found optimal as compared to Hadoop for most of the cases. Furthermore, Spark on YARN platform showed suboptimal results as compared to the case when it was executed in standalone mode. Some similar results were also observed by Zaharia et al. [32] on a 100 GB dataset record. Vellaipandiyan and Raja [33] demonstrated performance evaluation and comparison of Hadoop and Spark frameworks on resident's record dataset ranging from 100 GB to 900 GB of size. Spark scale of performance was relatively better when the dataset size was between small and medium size (100 GB–750 GB); afterwards, its performance declined as compared to Hadoop. The primary reason for the performance decline was evident as Spark cache size could not fit into the memory for the larger dataset. Taran et al. [34] quantified performance differences of Hadoop and Spark using WordCount dataset which was ranging from 100 KB to 1 GB. It was observed that Hadoop framework was five times faster than Spark when the evaluation was performed using a larger set of data sources. However, for the smaller tasks, Spark showed better performance results. However, the speed-up ratio was decreased for both databases with the growth of input dataset.

Gopalani and Arora [35] used K-Means algorithm on some small, medium, and large location datasets to compare Hadoop and Spark frameworks. The study results showed that Spark performed up to three times better than MapReduce for most of the cases. Bertoni et al. [36] performed the experimental evaluation of Apache Flink and Storm using large genomic dataset data on Amazon EC2 cloud. Apache Flink was superior to Storm while performing histogram and map operations while Storm outperformed Flink while genomic join application was deployed.

3.2. Fault Tolerance. Fault tolerance is the characteristic that enables a system to continue functioning in case of the failure of one or more components. High-performance computing applications involve hundreds of nodes that are interconnected to perform a specific task; failing a node should have zero or minimal effect on overall computation. The tolerance of a system, represented as $\mathrm{Tol}_{\mathrm{FT}}$, to meet its requirements after a disruption is the ratio of the time to complete tasks without observing any fault events to the overall execution time where some fault events were detected and the system state is reverted back to consistent state:

$$\mathrm{Tol}_{\mathrm{FT}} = \frac{T_x}{T_x + \sigma^2}, \tag{2}$$

where T_x is the estimated correct execution time obtained from a program run that is presumed to be fault-free, or by averaging the execution time from several application runs that produce a known correct output, and σ^2 represents the variance in a program's execution time due to the occurrence of fault events. For an HPC application that consists of a set of computationally intensive tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and since $\mathrm{Tol}_{\mathrm{FT}}$ for each individual task is computed as Tol_{τ_i}, then the overall application resilience, Tol, may be calculated as [37]

$$\mathrm{Tol} = \sqrt[n]{\mathrm{Tol}_{\tau_1} \cdot \mathrm{Tol}_{\tau_2} \cdot \ldots \cdot \mathrm{Tol}_{\tau_n}}. \tag{3}$$

Lu et al. [38] used StreamBench toolkit to evaluate performance and fault tolerance ability of Apache Spark, Storm, Spark, and Samza. It was found that, with the increased data size, Spark is much stable and fault-tolerant as compared to Storm but may be less efficient when compared to Samza. Furthermore, when compared in terms of handling large capacity of data, both Samza and Spark outperformed Storm. Gu and Li [39] used PageRank algorithm to perform a comparative experiment on Hadoop and Spark frameworks. It was observed that, for smaller datasets such as wiki-Vote and soc-Slashdot0902, Spark outperformed Hadoop with a much better margin. However, this speed-up result degraded with the growth of dataset, and for large datasets, Hadoop easily outperformed Spark. Furthermore, for massively large datasets, Spark was reported to be crashed with JVM heap exception while Hadoop still performed its task. Lopez et al. [40] evaluated throughput and fault tolerance mechanism of Apache Storm and Flink. The experiments were based on a threat detection system where Apache Storm demonstrated better throughput as compared to Flink. For fault tolerance,

different virtual machines were manually turned off to analyze the impact of node failures. Apache Flink used its internal subsystem to detect and migrate the failed tasks to other machines and hence resulted in very few message losses. On the other hand, Storm took more time as Zookeeper, involving some performance overhead, was responsible for reporting the state of nimbus and thereafter processing the failed task on other nodes.

3.3. Scalability. Scalability refers to the ability to accommodate large loads or change in size/workload by provisioning of resources at runtime. This can be further categorized as scale-up (by making hardware stronger) or scale-down (by adding additional nodes). One of the critical requirements of enterprises is to process large volumes of data in a timely manner to address high-value business problems. Dynamic resource scalability allows business entities to perform massive computation in parallel, thus reducing overall time, complexity, and effort. The definition of scalability comes from Amdahl's and Gustafson's laws [41]. Let W be the size of workload before the improvement of the system resources; the fraction of the execution workload that benefits from the improvement of system resources is α and the fraction concerning the part that would not benefit from improvement in the resources is $1 - \alpha$. When using an n-processor system, user workload is scaled to

$$\acute{W} = \alpha W + (1 - \alpha)\, nW. \tag{4}$$

The parallel execution time of a scaled workload $\acute{W}$ on n-processors is defined as scaled-workload speed-up $\acute{S}$ as shown in

$$\acute{S} = \frac{\acute{W}}{W} = \frac{\alpha W + (1 - \alpha)\, nW}{W}. \tag{5}$$

García-Gil et al. [42] performed scalability and performance comparison of Apache Spark and Flink using feature selection framework to assemble multiple information theoretic criteria into a single greedy algorithm. The ECBDL14 dataset was used to measure scalability factor for the frameworks. It was observed that Spark scalability performance was 4–10 times faster than Flink. Jakovits and Srirama [43] analyzed four MapReduce based frameworks including Hadoop and Spark for benchmarking partitioning around Medoids, Clustering Large Applications, and Conjugate Gradient linear system solver algorithms using MPI. All experiments were performed on 33 Amazon EC2 large instances cloud. For all algorithms, Spark performed much better as compared to Hadoop, in terms of both performance and scalability. Boden et al. [44] aimed to investigate the scalability with respect to both data size and dimensionality in order to demonstrate a fair and insightful benchmark that reflects the requirements of real-world machine learning applications. The benchmark was comprised of distributed optimization algorithms for supervised learning as well as algorithms for unsupervised learning. For supervised learning, they implemented machine learning algorithms by using Breeze library, while for unsupervised learning they chose the popular k-Means clustering algorithm in order to assess the scalability of the two resource management frameworks. The overall execution time of Flink was relatively low on the resource-constrained settings with a limited number of nodes, while Spark had a clear edge once enough main memory was available due to the addition of new computing nodes.

3.4. Machine Learning and Iterative Tasks Support. Big data applications are inherently complex in nature and usually involve tasks and algorithms that are iterative in nature. These applications have distinct cyclic nature to achieve the desired result by continually repeating a set of tasks until these cannot be substantially reduced further.

Spangenberg et al. [45] used real-world datasets, consisting of four algorithms, that is, WordCount, K-Means, PageRank, and relational query, to benchmark Apache Flink and Storm. It was observed that Apache Storm performs better in batch mode as compared to Flink. However, with the increasing complexity, Apache Flink had a performance advantage over Storm and thus it was better suited for iterative data or graph processing. Shi et al. [46] focused on analyzing Apache MapReduce and Spark for batch and iterative jobs. For smaller datasets, Spark resulted to be a better choice, but when experiments were performed on larger datasets, MapReduce turned out to be several times faster than Spark. For iterative operations such as K-Means, Spark turned out to be 1.5 times faster as compared to MapReduce in its first iteration, while Spark was more than 5 times faster in subsequent operations.

Kang and Lee [47] examined five resource management frameworks including Apache Hadoop and Spark with respect to performance overheads (disk input/output, network communication, scheduling, etc.) in supporting iterative computation. The PageRank algorithm was used to evaluate these performance issues. Since static data processing tends to be a more frequent operation than dynamic data as it is used in every iteration of MapReduce, it may cause significant performance overhead in case of MapReduce. On the other hand, Apache Spark uses read-only cached version of objects (resilient distributed dataset) which can be reused in parallel operations, thus reducing the performance overhead during iterative computation. Lee et al. [48] evaluated five systems including Hadoop and Spark over various workloads to compare against four iterative algorithms. The experimentation was performed on Amazon EC2 cloud. Overall, Spark showed the best performance when iterative operations were performed in main memory. In contrast, the performance of Hadoop was significantly poor as compared to other resource management systems.

3.5. Latency. Big data and low latency are strongly linked. Big data applications provide true value to businesses, but these are mostly time critical. If cloud computing has to be the successful platform for big data implementation, one of the key requirements will be the provisioning of high-speed network to reduce communication latency. Furthermore, big data frameworks usually involve centralized design where the scheduler assigns all tasks through a single node which may significantly impact the latency when the size of data is huge.

Let T_{elapsed} be the elapsed time between the start and finish time of a program in a distributed architecture, T_i be the effective execution time, and λ_i be the sum of total idle units of ith processor from a set of N processors. Then, the average latency, represented as $\lambda(W, N)$, for the size of workload W, is defined as the average amount of overhead time needed for each processor to complete the task:

$$\lambda(W, N) = \frac{\sum_{i=1}^{N} \left(T_{\text{elapsed}} - T_i + \lambda_i\right)}{N}. \quad (6)$$

Chintapalli et al. [49] conducted a detailed analysis of Apache Storm, Flink, and Spark streaming engines for latency and throughput. The study results indicated that, for high throughput, Flink and Storm have significantly lower latency as compared to Spark. However, Spark was able to handle high throughput as compared to other streaming engines. Lu et al. [38] proposed StreamBench benchmark framework to evaluate modern distributed stream processing frameworks. The framework includes dataset selection, data generation methodologies, program set description, workload suites design, and metric proposition. Two real-world datasets, AOL Search Data and CAIDA Anonymized Internet Traces Dataset, were used to assess performance, scalability, and fault tolerance aspects of streaming frameworks. It was observed that Storm's latency in most cases was far less than Spark's except in the case when the scales of the workload and dataset were massive in nature.

3.6. Security. Instead of the classical HPC environment, where information is stored in-house, many big data applications are now increasingly deployed on the cloud where privacy-sensitive information may be accessed or recorded by different data users with ease. Although data privacy and security issues are not a new topic in the area of distributed computing, their importance is amplified by the wide adoption of cloud computing services for big data platform. The dataset may be exposed to multiple users for different purposes which may lead to security and privacy risks.

Let N be a list of security categories that may be provided for security mechanism. For instance, a framework may use encryption mechanism to provide data security and access control list for authentication and authorization services. Let $\max(W_i)$ be the maximum weight that is assigned to the ith security category from a list of N categories and W_i be the reputation score of a particular resource management framework. Then, the framework security ranking score can be represented as

$$\text{Sec}_{\text{score}} = \frac{\sum_{i=1}^{N} W_i}{\sum_{i=1}^{N} \max\left(W_i\right)}. \quad (7)$$

Hadoop and Storm use Kerberos authentication protocol for computing nodes to provide their identity [50]. Spark adopts a password based shared secret configuration as well as Access Control Lists (ACLs) to control the authentication and authorization mechanisms. In Flink, stream brokers are responsible for providing authentication mechanism across multiple services. Apache Samza/Kafka provides no built-in security at the system level.

3.7. Dataset Size Support. Many scientific applications scale up to hundreds of nodes to process massive amount of data that may exceed over hundreds of terabytes. Unless big data applications are properly optimized for larger datasets, this may result in performance degradation with the growth of the data. Furthermore, in many cases, this may result in a crash of software, resulting in loss of time and money. We used the same methodology to collect big data support statistics as presented in earlier sections.

4. Discussion on Big Data Framework

Every big data framework has been evolved for its unique design characteristics and application-specific requirements. Based on the key factors and their empirical evidence to evaluate big data frameworks, as discussed in Section 3, we produce the summary of results of seven evaluation factors in the form of star ranking, $\text{Ranking}_{\text{RF}} \in [0, s]$, where s represents the maximum evaluation score. The general equation to produce the ranking, for each resource framework (RF), is given as

$$\text{Ranking}_{\text{RF}} = \left\| \frac{s}{\sum_{i=1}^{7} \sum_{j=1}^{N} \max\left(W_{ij}\right)} * \sum_{i=1}^{7} \sum_{j=1}^{N} W_{ij} \right\|, \quad (8)$$

where N is the total number of research studies for a particular set of evaluation metrics, $\max(W_{ij}) \in [0, 1]$ is the relative normalized weight assigned to each literature study based on the number of experiments performed, and W_{ij} is the framework test-bed score calculated from the experimentation results from each study.

Hadoop MapReduce has a clear edge on large-scale deployment and larger dataset processing. Hadoop is highly compatible and interoperable with other frameworks. It also offers a reliable fault tolerance mechanism to provide a failure-free mechanism over a long period of time. Hadoop can operate on a low-cost configuration. However, Hadoop is not suitable for real-time applications. It has a significant disadvantage when latency, throughput, and iterative job support for machine learning are the key considerations of application requirements.

Apache Spark is designed to be a replacement for batch-oriented Hadoop ecosystem to run-over static and real-time datasets. It is highly suitable for high throughput streaming applications where latency is not a major issue. Spark is memory intensive and all operations take place in memory. As a result, it may crash if enough memory is not available for further operations (before the release of Spark version 1.5, it was not capable of handling datasets larger than the size of RAM and the problem of handling larger dataset still persists in the newer releases with different performance overheads). Few research efforts, such as Project Tungsten, are aimed at addressing the efficiency of memory and CPU for Spark applications. Spark also lacks its own storage system so its integration with HDFS through YARN or Cassandra using Mesos is an extra overhead for cluster configuration.

Apache Flink is a true streaming engine. Flink supports both batch and real-time operations over a common runtime to fulfill the requirements of Lambda architecture. However,

it may also work in batch mode by stopping the streaming source. Like Spark, Flink performs all operations in memory, but in case of memory hog, it may also use disk storage to avoid application failure. Flink has some major advantages over Hadoop and Spark by providing better support for iterative processing with high throughput at the cost of low latency.

Apache Storm was designed to provide a scalable, fault tolerance, real-time streaming engine for data analysis, which Hadoop did for batch processing. However, the empirical evidence suggests that Apache Storm proved to be inefficient to meet the scale-up/scale-down requirements for real-time big data applications. Furthermore, since it uses microbath stream processing, it is not very efficient where continuous stream process is a major concern, nor does it provide a mechanism for simple batch processing. For fault tolerance, Storm uses Zookeeper to store the state of the processes which may involve some extra overhead and may also result in message loss. On the other hand, Storm is an ideal solution for near-real-time application processing where workload could be processed with a minimal delay with strict latency requirements.

Apache Samza, in integration Kafka, provides some unique features that are not offered by other stream processing engines. Samza provides a powerful check-pointing based fault tolerance mechanism with minimal data loss. Samza jobs can have high throughput with low latency when integrated with Kafka. However, Samza lacks some important features as data processing engine. Furthermore, it offers no built-in security mechanism for data access control.

To categorize the selection of best resource engine based on a particular set of requirements, we use the framework proposed by Chung et al. [51]. The framework provides a layout for matching, ranking, and selecting a system based on some particular requirements. The matching criterion is based on user goals which are categorized as soft and hard goals. Soft goals represent the nonfunctional requirements of the system (such as security, fault tolerance, and scalability) while hard goals represent the functional aspects of the system (such as machine learning and data size support). The relationship between the system components and the goals can be ranked as very positive (++), positive (+), negative (−), and very negative (−−). Based on the evidence provided from the literature, the categorization of major resource management frameworks is presented in Figure 7.

5. Related Work

Our research work differs from other efforts because the subject goal and object of study are not identical, as we provide an in-depth comparison of popular resource engines based on empirical evidence from existing literature. Hesse and Lorenz [8] conducted a conceptual survey on stream processing systems. However, their discussion was focused on some basic differences related to real-time data processing engines. Singh and Reddy [9] provided a thorough analysis of big data analytic platforms that included peer-to-peer networks, field programmable gate arrays (FPGA), Apache Hadoop ecosystem, high-performance computing (HPC) clusters, multicore CPU, and graphics processing unit (GPU). Our case is different here as we are particularly interested in big data processing engines. Finally, Landset et al. [10] focused on machine learning libraries and their evaluation based on ease of use, scalability, and extensibility. However, our work differs from their work as the primary focuses of both studies are not identical.

Chen and Zhang [11] discussed big data problems, challenges, and associated techniques and technologies to address these issues. Several potential techniques including cloud computing, quantum computing, granular computing, and biological computing were investigated and the possible opportunities to explore these domains were demonstrated. However, the performance evaluation was discussed only on theoretical grounds. A taxonomy and detailed analysis of the state of the art in big data 2.0 processing systems were presented in [12]. The focus of the study was to identify current research challenges and highlight opportunities for new innovations and optimization for future research and development. Assunção et al. [13] reviewed multiple generations of data stream processing frameworks that provide mechanisms for resource elasticity to match the demands of stream processing services. The study examined the challenges associated with efficient resource management decisions and suggested solutions derived from the existing research studies. However, the study metrics are restricted to the elasticity/scalability aspect of big data streaming frameworks.

As shown in Table 3, our work differs from the previous studies which focused on the classification of resource management frameworks on theoretical grounds. In contrast to the earlier approaches, we classify and categorize big data resource management frameworks based on empirical grounds, derived from multiple evaluation/experimentation studies. Furthermore, our evaluation/ranking methodology is based on a comprehensive list of study variables which were not addressed in the studies conducted earlier.

6. Conclusions and Future Work

There are a number of disruptive and transformative big data technologies and solutions that are rapidly emanating and evolving in order to provide data-driven insight and innovation. The primary object of the study was how to classify popular big data resource management systems. This study was also aimed at addressing the selection of candidate resource provider based on specific big data application requirements. We surveyed different big data resource management frameworks and investigated the advantages and disadvantages for each of them. We carried out the performance evaluation of resource management engines based on seven key factors and each one of the frameworks was ranked based on the empirical evidence from the literature.

6.1. Observations and Findings. Some key findings of the study are as follows:

(i) In terms of processing speed, Apache Flink outperforms other resource management frameworks for small, medium, and large datasets [30, 36]. However, during our own set of experiments on Amazon EC2

Table 3: Comparison and application areas of related research studies.

Study reference	Data model	Resource frameworks	Study features	Evaluation/ranking methodology
[8]	Data stream processing systems	Storm, Flink, Spark, Samza	A brief comparison of resource frameworks	✘
[9]	Batch and stream processing systems	Horizontal scaling systems, such as peer-to-peer, MapReduce/MPI, and Spark, and vertical scaling systems, such as CUDA and HDL	Comparison of horizontal and vertical scaling systems	Theoretical comparison of resource frameworks
[10]	Batch and stream processing engines	MapReduce, Spark, Flink, and Storm as well as machine learning libraries	Machine learning libraries and their evaluation mechanism	Performance comparison with respect to machine learning toolkits
[11]	Batch and stream processing frameworks	Hadoop, Storm, and other big data frameworks	In-depth analysis of big data opportunities and challenges	✘
[12]	Batch and stream processing frameworks	Hadoop, Spark, Storm, Flink, and Tez as well as SQL, Graph, and bulk synchronous parallel model	Analysis of current open research challenges in the field of big data and the promising directions for future research	✘
[13]	Stream processing engines	Apache Storm, S4, Flink, Samza, Spark Streaming, and Twitter Heron	Classification of elasticity metrics for resource allocation strategies that meet the demands of stream processing services	Evaluation of elasticity/scaling metrics for stream processing systems

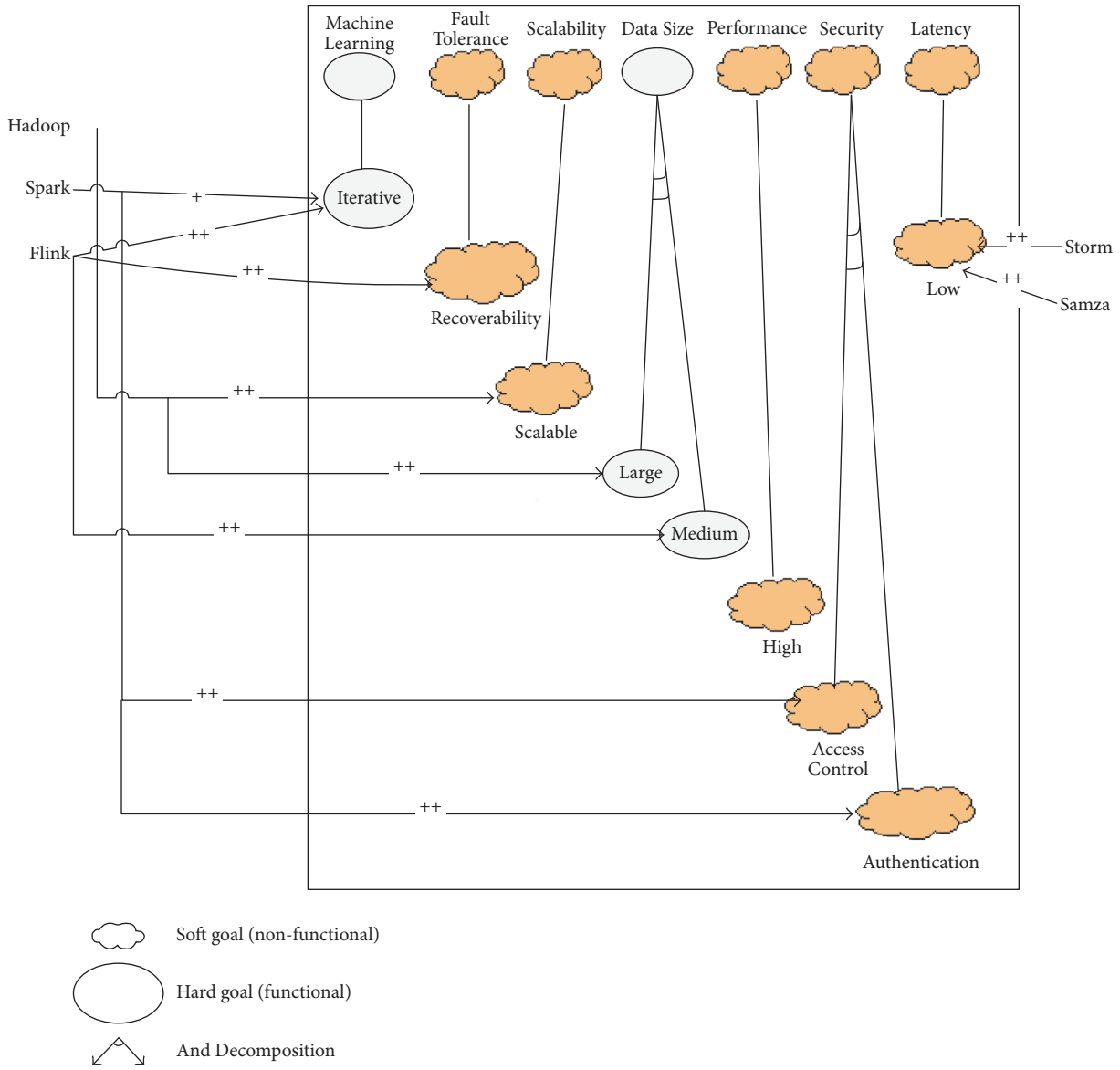

Figure 7: Categorization of resource engines based on key big data application requirements.

cluster with varied task managers settings (1–4 task managers per node), Flink failed to complete custom smaller size JVM dataset jobs due to inefficient memory management of Flink memory manager. We could not find any reported evidence of this particular use case in these relevant literature studies. It seems that most of the performance evaluation studies employed standard benchmarking test sets where dataset size was relatively large and hence this particular use case was not reported in these studies. Further research effort is required to elucidate the underlying specific of factors under this particular case.

(ii) Big data applications usually involve massive amount of data. Apache Spark supports necessary strategies for fault tolerance mechanism, but it has been reported to crash on larger datasets. Even during our experimentations, Apache Spark version 1.6 (selected due to the compatibility reasons with earlier researches) crashed on several occasions when the dataset was larger than 500 GB. Although Spark has been ranked higher in terms of fault tolerance with the increase of data scale in several studies [38, 52], it has limitations in handling larger typical big data dataset applications and hence such studies cannot be generalized.

(iii) Spark MLlib and Flink-ML offer a variety of machine learning algorithms and utilities to exploit distributed and scalable big data applications. Spark MLlib outperforms Flink-ML in most of the machine learning use cases [42], except in the case when repeated passes are performed on unchanged data. However, this performance evaluation may further be investigated

as research studies such as [53], reported differently where Flink outperformed Spark on sufficiently large cases.

(iv) For graph processing algorithms such as PageRank, Flink uses Gelly library that provides native closed-loop iteration operators, making it a suitable platform for large-scale graph analytics. Spark, on the other hand, uses GraphX library that has much longer pre-processing time to build graph and other data structures, making its performance worse as compared to Apache Flink. Apache Flink has been reported to obtain the best results for graph processing datasets (3x–5x in [54] and 2x-3x in [45]) as compared to Spark. However, some studies such as [55] reported Spark to be 1.7x faster than Flink for large graph processing. Such inconsistent behavior may be further investigated in the future research studies.

6.2. Future Work. In the area of big data, there is still a clear gap that requires more effort from the research community to build in-depth understanding of performance characteristics of big data resource management frameworks. We consider this study a step towards enlarging our knowledge to understand the big data world and provide an effort towards the direction of improving the state of the art and achieving the big vision on the big data domain. In earlier studies, a clear ranking cannot be established as the study parameters were mostly limited to a few issues such as throughput, latency, and machine learning. Furthermore, further investigation is required on resource engines such as Apache Samza in comparison with other frameworks. In addition, research effort needs to be carried out in several areas such as data organization, platform specific tools, and technological issues in big data domain in order to create next-generation big data infrastructures. The performance evaluation factors might also vary among systems depending on the used algorithms. As future work, we plan to benchmark these popular resource engines for meeting resource demands and requirements for different scientific applications. Moreover, a scalability analysis could be done. Particularly, the performance evaluation of adding dynamic worker nodes and the resulting performance analysis is of peculiar interest. Additionally, further research can be carried out in order to evaluate performance aspects with respect to resource competition between jobs (on different research schedulers such as YARN and Mesos) and the fluctuation of available computing resources. Finally, most of the experimentations in earlier studies were performed using standard parameter configurations; however, each resource management framework offers domain specific tweaks and configuration optimization mechanisms for meeting application-specific requirements. The development of a benchmark suite that aims to find maximum throughput based on configuration optimization would be an interesting direction of future research.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] "Big Data in the Cloud: Converging Technologies, (August 11)," https://www.intel.com/content/dam/www/public/emea/de/de/documents/product-briefs/big-data-cloud-technologies-brief.pdf.

[2] Q. He, J. Yan, R. Kowalczyk, H. Jin, and Y. Yang, "Lifetime service level agreement management with autonomous agents for services provision," *Information Sciences*, vol. 179, no. 15, pp. 2591–2605, 2009.

[3] O. Rana, M. Warnier, T. B. Quillinan, and F. Brazier, "Monitoring and reputation mechanisms for service level agreements," in *5th International Workshop on Grid Economics and Business Models, GECON '08*, vol. 5206 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, pp. 125–139, 2008.

[4] N. Yuhanna and M. Gualtieri, "The Forrester Wave™: Big Data Hadoop Cloud Solutions, Q2 2016 Elasticity, Automation, and Pay-As-You-Go Compel Enterprise Adoption of Hadoop in the Cloud, (June 20)," https://ncmedia.azureedge.net/ncmedia/2016/05/The_Forrester_Wave__Big_D.pdf.

[5] R. Arora, "An introduction to big data, high performance computing, high-throughput computing, and Hadoop," *Conquering Big Data with High Performance Computing*, pp. 1–12, 2016.

[6] F. Pop, J. Kołodziej, and B. D. Martino, *Resource Management for Big Data Platforms Algorithms, Modelling, and High- Performance Computing Techniques*, Springer, 2016.

[7] M. Gribaudo, M. Iacono, and F. Palmieri, "Performance modeling of big data-oriented architectures," in *Resource Management for Big Data Platforms*, Computer Communications and Networks, pp. 3–34, Springer International Publishing, Cham, 2016.

[8] G. Hesse and M. Lorenz, "Conceptual survey on data stream processing systems," in *Proceedings of the 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 797–802, Melbourne, Australia, December 2015.

[9] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of Big Data*, vol. 2, Article ID 8, 2014.

[10] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem," *Journal of Big Data*, vol. 2, no. 1, Article ID 24, 2015.

[11] C. L. P. Chen and C. Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: a survey on Big Data," *Information Sciences*, vol. 275, pp. 314–347, 2014.

[12] F. Bajaber, R. Elshawi, O. Batarfi, A. Altalhi, A. Barnawi, and S. Sakr, "Big data 2.0 processing systems: taxonomy and open challenges," *Journal of Grid Computing*, vol. 14, no. 3, pp. 379–405, 2016.

[13] M. D. d. Assunção, A. d. S. Veith, and R. Buyya, "Distributed data stream processing and edge computing: a survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, 2018.

[14] "Big data working group big data taxonomy, 2014".

[15] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. M. Nguifo, "An experimental survey on big data frameworks," *Future Generation Computer Systems*, 2018.

[16] H.-K. Kwon and K.-K. Seo, "A fuzzy AHP based multi-criteria decision-making model to select a cloud service," *International Journal of Smart Home*, vol. 8, no. 3, pp. 175–180, 2014.

[17] A. Rezgui and S. Rezgui, "A stochastic approach for virtual machine placement in volunteer cloud federations," in *Proceedings of the 2nd IEEE International Conference on Cloud Engineering, IC2E 2014*, pp. 277–282, Boston, MA, USA, March 2014.

[18] M. Salama, A. Zeid, A. Shawish, and X. Jiang, "A novel QoS-based framework for cloud computing service provider selection," *International Journal of Cloud Applications and Computing*, vol. 4, no. 2, pp. 48–72, 2014.

[19] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid computing-where hpc meets grid and cloud computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 440–453, 2011.

[20] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[21] S. Benedict, "Performance issues and performance analysis tools for HPC cloud applications: a survey," *Computing*, vol. 95, no. 2, pp. 89–108, 2013.

[22] E. C. Inacio and M. A. R. Dantas, "A survey into performance and energy efficiency in HPC, cloud and big data environments," *International Journal of Networking and Virtual Organisations*, vol. 14, no. 4, pp. 299–318, 2014.

[23] N. Yuhanna and M. Gualtieri, "Elasticity, Automation, and Pay-As-You-Go Compel Enterprise Adoption Of Hadoop in the Cloud, Q2, 2016".

[24] S. Ullah, M. D. Awan, and M. S. Khiyal, "A price-performance analysis of EC2, google compute and rackspace cloud providers for scientific computing," *Journal of Mathematics and Computer Science*, vol. 16, no. 02, pp. 178–192, 2016.

[25] J. Hurwitz, A. Nugent, F. Halper, and M. Kaufman, *Big Data for Dummies*, John Wiley Sons, 2013.

[26] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, 2013.

[27] D. Wu, S. Sakr, and L. Zhu, "Big data programming models," in *Handbook of Big Data Technologies*, pp. 31–63, 2017.

[28] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera, "Big data preprocessing: methods and prospects," *Big Data Analytics*, vol. 1, no. 1, 2016.

[29] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*, pp. 1–14, ACM, Melbourne, Australia, November 2010.

[30] J. Veiga, R. R. Exposito, X. C. Pardo, G. L. Taboada, and J. Tourifio, "Performance evaluation of big data frameworks for large-scale data analytics," in *Proceedings of the 4th IEEE International Conference on Big Data, Big Data 2016*, pp. 424–431, December 2016.

[31] I. Mavridis and H. Karatza, "Log file analysis in cloud with Apache Hadoop and Apache Spark," in *Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)*, Poland, 2015.

[32] M. Zaharia, M. Chowdhury, and T. Das, "Fast and interactive analytics over Hadoop data with Spark," *USENIX Login*, vol. 37, no. 4, pp. 45–51, 2012.

[33] S. Vellaipandiyan and P. V. Raja, "Performance evaluation of distributed framework over YARN cluster manager," in *Proceedings of the 2016 IEEE International Conference on Computational Intelligence and Computing Research, ICCIC 2016*, December 2016.

[34] V. Taran, O. Alienin, S. Stirenko, Y. Gordienko, and A. Rojbi, "Performance evaluation of distributed computing environments with Hadoop and Spark frameworks," in *Proceedings of the 2017 IEEE International Young Scientists' Forum on Applied Physics and Engineering (YSF)*, pp. 80–83, Lviv, Ukraine, October 2017.

[35] S. Gopalani and R. Arora, "Comparing Apache Spark and Map Reduce with performance analysis using K-means," *International Journal of Computer Applications*, vol. 113, no. 1, pp. 8–11, 2015.

[36] M. Bertoni, S. Ceri, A. Kaitoua, and P. Pinoli, "Evaluating cloud frameworks on genomic applications," in *Proceedings of the 3rd IEEE International Conference on Big Data, IEEE Big Data 2015*, pp. 193–202, November 2015.

[37] S. Hukerikar, R. A. Ashraf, and C. Engelmann, "Towards new metrics for high-performance computing resilience," in *Proceedings of the 7th Fault Tolerance for HPC at eXtreme Scale Workshop, FTXS 2017*, pp. 23–30, June 2017.

[38] R. Lu, G. Wu, B. Xie, and J. Hu, "Stream bench: towards benchmarking modern distributed stream computing frameworks," in *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2014*, pp. 69–78, December 2014.

[39] L. Gu and H. Li, "Memory or time: performance evaluation for iterative operation on Hadoop and Spark," in *Proceedings of the 15th IEEE International Conference on High Performance Computing and Communications, HPCC 2013 and 11th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, EUC 2013*, pp. 721–727, November 2013.

[40] M. A. Lopez, A. G. P. Lobato, and O. C. M. B. Duarte, "A performance comparison of open-source stream processing platforms," in *Proceedings of the 59th IEEE Global Communications Conference, GLOBECOM 2016*, December 2016.

[41] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532-533, 1988.

[42] D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera, "A comparison on scalability for batch big data processing on Apache Spark and Apache Flink," *Big Data Analytics*, vol. 2, no. 1, 2017.

[43] P. Jakovits and S. N. Srirama, "Evaluating Mapreduce frameworks for iterative scientific computing applications," in *Proceedings of the 2014 International Conference on High Performance Computing and Simulation, HPCS 2014*, pp. 226–233, July 2014.

[44] C. Boden, A. Spina, T. Rabl, and V. Markl, "Benchmarking data flow systems for scalable machine learning," in *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond, BeyondMR 2017*, May 2017.

[45] N. Spangenberg, M. Roth, and B. Franczyk, *Evaluating New Approaches of Big Data Analytics Frameworks*, vol. 208 of *Lecture Notes in Business Information Processing*, Publisher Name Springer, Cham, 2015.

[46] J. Shi, Y. Qiu, U. F. Minhas et al., "Clash of the titans: MapReduce vs. Spark for large scale data analytics," in *Proceedings of the 41st International Conference on Very Large Data Bases*, pp. 2110–2121, Kohala Coast, HI, USA, 2015.

[47] M. Kang and J. Lee, "A comparative analysis of iterative MapReduce systems," in *Proceedings of the the Sixth International Conference*, pp. 61–64, Jeju, Repbulic of Korea, October 2016.

[48] H. Lee, M. Kang, S.-B. Youn, J.-G. Lee, and Y. Kwon, "An experimental comparison of iterative MapReduce frameworks,"

in *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016*, pp. 2089–2094, October 2016.

[49] S. Chintapalli, D. Dagit, B. Evans et al., "Benchmarking streaming computation engines: Storm, Flink and Spark streaming," in *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2016*, pp. 1789–1792, May 2016.

[50] X. Zhang, C. Liu, S. Nepal, W. Dou, and J. Chen, "Privacy-preserving layer over MapReduce on cloud," in *Proceedings of the 2nd International Conference on Cloud and Green Computing, CGC 2012, Held Jointly with the 2nd International Conference on Social Computing and Its Applications, SCA 2012*, pp. 304–310, chn, November 2012.

[51] L. Chung, B. A. Nixon, and E. Yu, "Using nonfunctional requirements to systematically select among alternatives in architectural design," in *Proceedings of the 1st International Workshop on Architectures for Software Systems*, pp. 31–43, 1995.

[52] S. Qian, G. Wu, J. Huang, and T. Das, "Benchmarking modern distributed streaming platforms," in *Proceedings of the 2016 IEEE International Conference on Industrial Technology (ICIT)*, pp. 592–598, Taipei, Taiwan, March 2016.

[53] F. Dambreville and A. Toumi, "Generic and massively concurrent computation of belief combination rules," in *Proceedings of the the International Conference on Big Data and Advanced Wireless Technologies*, pp. 1–6, Blagoevgrad, Bulgaria, November 2016.

[54] R. W. Techentin, M. W. Markland, R. J. Poole, D. R. H. C. R. Haider, and B. K. Gilbert, "Page rank performance evaluation of cluster computing frameworks on Cray Urika-GX supercomputer," *Computer Science and Engineering*, vol. 6, pp. 33–38, 2016.

[55] O.-C. Marcu, A. Costan, G. Antoniu, and M. S. Pérez-Hernández, "Spark versus flink: understanding performance in big data analytics frameworks," in *Proceedings of the 2016 IEEE International Conference on Cluster Computing, CLUSTER 2016*, pp. 433–442, Taipei, Taiwan, September 2016.