BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 04/10 pp154–169
Volume 6, Number 2, June 2023
DOI: 10.26599/BDMA.2022.9020014

Survey of Distributed Computing Frameworks for Supporting Big Data Analysis

Xudong Sun, Yulin He, Dingming Wu, and Joshua Zhexue Huang*

Abstract: Distributed computing frameworks are the fundamental component of distributed computing systems. They provide an essential way to support the efficient processing of big data on clusters or cloud. The size of big data increases at a pace that is faster than the increase in the big data processing capacity of clusters. Thus, distributed computing frameworks based on the MapReduce computing model are not adequate to support big data analysis tasks which often require running complex analytical algorithms on extremely big data sets in terabytes. In performing such tasks, these frameworks face three challenges: computational inefficiency due to high I/O and communication costs, non-scalability to big data due to memory limit, and limited analytical algorithms because many serial algorithms cannot be implemented in the MapReduce programming model. New distributed computing frameworks need to be developed to conquer these challenges. In this paper, we review MapReduce-type distributed computing frameworks that are currently used in handling big data and discuss their problems when conducting big data analysis. In addition, we present a non-MapReduce distributed computing framework that has the potential to overcome big data analysis challenges.

Key words: distributed computing frameworks; big data analysis; approximate computing; MapReduce computing model

1 Introduction

In the era of big data, an overwhelming amount of data of various types is generated at all times from different channels, such as social networks, the Internet of Things, business transactions, finance networks, and personal media[1]. As a consequence, the explosive growth of global data has led to a fast increase in scales of accumulated data in data centers all over the world[2, 3]. Through an analysis of big data, valuable information and knowledge can be obtained, which benefits people from all walks of life and government and industry decision-makers[4, 5]. In this scenario, distributed computing plays the most important role in storing, processing, and analyzing big data.

Distributed computing frameworks are the fundamental component of distributed computing systems. Using the divide-and-conquer strategy[6, 7], they provide an essential way to support the efficient processing of big data on clusters or cloud. In these frameworks, a big data file is partitioned into a number of small files called data block files, which are stored in a distributed fashion on the nodes of a cluster and managed by using a distributed file system such as Google File System (GFS)[8] and Hadoop Distributed File System (HDFS)[9–11].

To process big data files on a cluster made of independent servers as nodes based on the shared-nothing architecture, local data block files are processed

• Xudong Sun, Yulin He, Dingming Wu, and Joshua Zhexue Huang are with College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China. E-mail: sunxudong2016@email.szu.edu.cn; yulinhe@szu.edu.cn; dingming@szu.edu.cn; zx.huang@szu.edu.cn.
• Yulin He and Joshua Zhexue Huang are also with Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen 518107, China.
* To whom correspondence should be addressed.
Manuscript received: 2022-06-15; accepted: 2022-06-28

first on the local nodes in parallel, and then the local results are integrated into the global results[9]. Currently, as an industrial quasi-standard distributed computing platform, Apache Hadoop is widely used for handling big data. MapReduce is widely adopted as a general programming/computing model for implementing/executing distributed algorithms that can run on clusters and cloud. Other systems, such as Spark, Storm, and Flink, are also integrated into the Apache platform to provide different sets of functions for processing big data of different types and enrich the big data technology stack to support broad applications.

Today, the size of big data files is increasing at a pace that is much faster than the increase in the capacity of big data platforms to process and analyze them. When big data files exceed certain sizes, e.g., several gigabytes or a few terabytes, the big data platforms currently used in many organizations are inadequate for big data analysis tasks due to the hardware and software limits of big data platforms. These systems have three challenges: they are computationally inefficient due to high I/O and communication costs, unscalable to big data due to memory limits, and lack available analytical algorithms because many serial algorithms cannot be implemented in a MapReduce fashion.

Many efforts are being devoted to developing new technologies to tackle these challenging problems in big data analysis. An innovative approach to solving big data analysis problems was recently proposed in Refs. [12, 13]. In this approach, a random sample partition (RSP) data model is defined to represent a distributed big data file as a set of RSP data blocks, which are the random samples of big data files, and the RSP data blocks are saved in an HDFS file. When the big data file is analyzed, a set of RSP data blocks are randomly selected[14], and each data block is analyzed in each local node to produce the block-level result. Finally, all block-level results are integrated into the final result which represents the approximate estimate of the result that would be computed from the entire data file. The RSP approach is a promising technology for big data analysis because of its high computational efficiency, high scalability to big data, and non-MapReduce way of executing analytical algorithms.

In this paper, we first give a brief overview of available technologies and systems for handling big data. Then, we review three MapReduce-type distributed computing frameworks: Hadoop MapReduce, Haloop, and Spark, and discuss their problems in supporting big data analysis. After that, we present a non-MapReduce distributed computing framework enabled by the RSP data model[13]. The new non-MapReduce distributed computing framework has the following advantages: it significantly reduces communication cost; it is scalable to big data; it is able to run serial algorithms; and it supports approximate computing for big data analysis. Therefore, it is a promising approach to overcome big data analysis challenges.

The remainder of this paper is organized as follows. Section 2 gives a broad overview of technologies for handling big data. Section 3 discusses MapReduce-type distributed computing frameworks in the context of big data analysis. Section 4 is devoted to the non-MapReduce distributed computing framework. Section 5 presents the evaluations of current data analysis systems over nine parameters, which indicate the big data analysis ability. Finally, Section 6 concludes the survey.

2 Distributed Technologies for Handling Big Data

In this section, we present a brief overview of the available technologies for handling big data, including the system architectures of clusters and high-performance computing (HPC), distributed file systems, MapReduce programming model, middleware for distributed computing, Hadoop, and Spark, as well as other packages, such as Flink, Storm, Hive, and Pig Latin, which form the distributed computing technology stack for handling big data.

2.1 Clusters and HPC

Storing and processing big data files require a large amount of hardware resources: hard disks, memory, CPU/GPU, communication network bandwidth, and I/O speed. Physically and economically, using one big server to handle big data computing tasks is no longer an option. Therefore, clusters of servers are built based on the divide-and-conquer strategy to enable parallel and distributed computing for big data. Clusters have become ubiquitous for storing, processing, and analyzing big data.

Two mainstream system architectures are available for building parallel and distributed computing systems: HPC and cluster computing. The first one is used to build ultra-HPC systems called supercomputers, which integrate thousands of computing nodes and millions of CPU/GPU cores to perform various operations

in parallel[15–18] to solve computationally intensive computing tasks such as weather forecasting, high-energy physics, genetic analysis, and atmospheric science[19,20].

A cluster computing system connects multiple computing nodes through a network to form a cluster of computers, which complete segmented computing tasks through distributed computing and communication between nodes[21]. Compared with a supercomputer, a computing cluster can be easily built at a low cost. Cloud computing virtualizes the storage, memory, CPU, and network bandwidth resources to optimize resource utilization and provides users with virtual and transparent computing services for data management and processing[22–24].

Currently, the clusters and cloud are widely used in big data processing and analysis applications due to their availability, low cost, and usability. Although many studies have been conducted in the research community on using HPC systems in machine learning and data mining, these systems are not good choices in real applications because they are expensive to build and difficult to use. Although many machine learning algorithms have been parallelized to run on HPC systems, they are not scalable to big data, and some do not have a high performance[25–31]. In this paper, we will focus on the clusters of distributed computing in big data analysis.

2.2 Distributed file systems

With the use of the divide-and-conquer strategy in distributed computing, a big data file is partitioned into a number of small files, called data blocks, which are stored in a distributed manner on the disks of cluster nodes to improve I/O performance. A big data file stored in this way is called a distributed data file, which is managed on the cluster with a distributed file system[32,33], e.g., GFS[8], HDFS[34], Taobao file system (TFS)[35,36], and FastDFS[37]. The distributed file systems provide an important technical foundation for big data analysis[38].

GFS is a Linux-based distributed file system developed by Google to meet the needs of individual companies[39]. TFS is a high-availability, high-performance distributed file system developed by Taobao to meet the storage requirements of unstructured small files (usually no more than 1 MB). FastDFS is a lightweight open-source distributed file system that is especially suitable for online services using files as the carrier[37].

HDFS, which was developed in the Apache Hadoop project, was designed to overcome the challenges of distributed data processing in a large-scale cluster. It is a fault-tolerant data storage file system that runs on commodity hardware and is suitable for handling big data. Therefore, HDFS is widely adopted in the industry for big data processing and analysis.

2.3 MapReduce

The distributed file system divides big data files, and the MapReduce programming model divides the algorithm into pieces, which can run on the data blocks in a distributed fashion to gain a good computing performance. MapReduce was first developed by Google and later adopted in the Apache Hadoop project to rewrite serial algorithms in a MapReduce style, which can run efficiently on a cluster[40,41].

A MapReduce program is made up of two basic operations: Map and Reduce[42]. The Map operation is performed on the data blocks of a node to generate a local result. The Map operation is executed on the multiple nodes in parallel and independently to generate the local results on the corresponding nodes and gain a high computing performance[43]. The Reduce operation operates on the local results to generate a global result. In doing so, all local results are transferred to the nodes that perform the Reduce operation. This stage generates a high data communication cost due to the data shuffling and transformation among the nodes. After all local results are fetched to the Reduce nodes, a global result is generated by the Reduce operation[44].

If a data computing task can be completed in one pair of Map and Reduce operations, e.g., counting the word frequencies from a large number of web pages, then the MapReduce program can run efficiently over a big data file, because it can be executed in a large-scale cluster with thousands of nodes. However, if an iterative algorithm is rewritten as a sequence of Map and Reduce pairs, then the algorithm will not be able to run efficiently on a big distributed dataset due to the I/O, communication, and computing costs[45]. Therefore, the MapReduce programming model is not a choice for big data analysis[46]. We will discuss the reasons in Section 3.

2.4 Middleware

The distributed file system HDFS and MapReduce programming/computing model, the two core technologies of cluster computing for processing

distributed big data. They enable the divisions of both big data sets and algorithms for distributed computing based on the divide-and-conquer strategy. However, they cannot work without the support of other technologies. Cluster middleware was developed to support the execution of MapReduce programs and schedule the use of resources to optimize the executions.

The most popular middleware is YARN[47], which is a framework for resource management and task scheduling[48,49]. When a job is submitted, YARN will allocate suitable computing resources to execute the sequence of subtasks in the job (often a sequence of pairs of Map and Reduce operations on HDFS data blocks) and then recycle resources until the job is complete. YARN uses one of the strategies, namely, first in and first out, fair, or capacity scheduler, to schedule multiple jobs.

YARN is a distributed middleware used on various platforms such as Hadoop and Spark. It enables the platform to allocate and monitor computing resources and efficiently promote the execution of distributed computing tasks. In addition, Spark can also be deployed with Mesos and standalone middleware[50].

2.5 Hadoop platform

Apache Hadoop is an open-source software platform that manages data storage and processing for big data applications. The Hadoop platform integrates a number of software packages that provide different functions for different applications. For example, HDFS is software for distributed file management, and Hadoop MapReduce is a software framework for implementing MapReduce algorithms for processing large distributed datasets on clusters. The Hadoop platform enables these software packages to run in a computing cluster or cloud in coherence to complete the tasks of different applications[51–53].

Hadoop uses a master-slave architecture to efficiently store and process large amounts of data. In the process of executing a computing task, the master node assigns subtasks to the slave nodes. The slave nodes store the local data block files and execute the local calculations on the local data[54]. All intermediate results generated during this period can be cached by HDFS. Theoretically, Hadoop can process unlimited data but also incurs severe I/O costs.

2.6 Spark

Apache Spark, which was originally developed at the University of California, Berkeley, is an open-source unified engine for large-scale data processing. Unlike Hadoop MapReduce, Spark caches all intermediate results in the data structure of a Resilient Distributed Dataset (RDD) in the memory to reduce I/O costs[1]. It also introduces a directed acyclic graph (DAG) task segmentation mechanism to operate on RDD in a way similar to MapReduce. Spark in-memory computing is much faster than Hadoop, which makes Spark the current mainstream batch big data analysis platform[55–57]. The pros and cons of Spark in big data analysis will be discussed in the next section.

The in-memory computing technology used by Spark reduces the I/O overhead because the data are loaded into the memory for computing[58]. However, with the rapid expansion of the data scale, a memory bottleneck is unavoidable if the data size is larger than the memory. In this sense, data scalability is limited by the available memory, although it can be improved by adding nodes and more memory at a high cost. In addition, because many algorithms cannot be rewritten as MapReduce operations, MLlib and machine learning (ML) libraries in Spark have only limited algorithms available[59]. Nevertheless, as an efficient data processing engine, Spark is widely used in many applications.

2.7 Other packages

Hadoop and Spark, used as batch processing platforms, play an important role in offline big data processing and analysis. Other packages have been developed for handling big data of different types. We briefly review four packages for streaming data handling and database, as well as data warehouse applications.

When handling data streams, stream processing platforms, such as Flink and Storm, have performance advantages[60]. These stream processing platforms also use the divide-and-conquer strategy to divide data streams into pieces that are processed concurrently on each resource unit[61,62]. Apache Flink is a distributed processing engine for stateful computations over unbounded and bounded data streams. It runs in cluster environments and performs computations at in-memory speed and at any scale. Flink has a Hadoop compatibility package to wrap functions implemented in Hadoop MapReduce and embeds them in Flink programs. In this way, a bounded data stream can be handled as a batch of processed data in Hadoop MapReduce[63,64]. Storm is more focused on low-latency real-time computing[65,66] and does not run on the Hadoop platform.

For database query and data warehouse applications,

Hadoop platform provides Hive and Pig Latin packages, which use MapReduce programs to process data queries over big data files in HDFS[67,68]. Both have a high-level language to assist users in generating jobs without knowing the details of HDFS and MapReduce. The difference between them is that Hive is mainly used for structured data[69], while Pig Latin is a more lightweight data flow scripting language that is more suitable for unstructured data[70].

2.8 Stack for big data platform

Figure 1 shows a technology stack of distributed computing for handling big data. The technologies are summarized according to the functions of common components in the big data platform stack[71,72].

The data acquisition and transmission layer are the data sources for big data analysis. The purpose is to provide the functions for data collection and preprocessing and to facilitate the preservation of the data storage layer. For example, Flume is a distributed and highly available data collection, aggregation, and movement tool. Canal is an open-source tool used to extract data from MySQL databases and output it to any target storage. Kafka and Sqoop are used for the same purpose in different data environments[73,74].

The data storage layer includes the tools for data storage and management. For example, HBase is Hadoop's distributed and scalable big data store[75] for data management. This layer also includes the various distributed file systems (HDFS, GFS, etc.) and databases (Redis, TiDB, etc.) suitable for different data scenarios[76,77].

The analysis layer includes several distributed computing frameworks, such as Hadoop, Spark, Storm, and Hive, which are used for big data processing and analysis. Some parallel algorithm libraries are also included, such as Mahout and SparkML[78,79].

The online analysis processing layer contains tools for applications of business intelligence. These tools allow users to quickly read, observe, and explore multidimensional big data from different angles to assist in decision-making. Examples are Druid, Kylin, Impala, and Presto[80,81].

The resource management layer shows the general tools for the resource allocation and scheduling of various distributed computing tasks on distributed computing platforms. Examples are YARN[82], Kubernetes, and Mesos[83].

Ambari and Zookeeper are two commonly used service software that supports the distributed computing framework installation and deployment[84,85]. They provide the functions of monitoring, management, and configuration of most components and computing nodes.

3 Distributed Computing Frameworks for Big Data Analysis

In the previous section, we reviewed the technologies for handling big data in cluster systems or cloud. In this section, we review the distributed computing frameworks for supporting big data analysis tasks. We particularly focus on the analysis tasks that require complex iterative algorithms. These algorithms iterate on the dataset many times to obtain a final result.

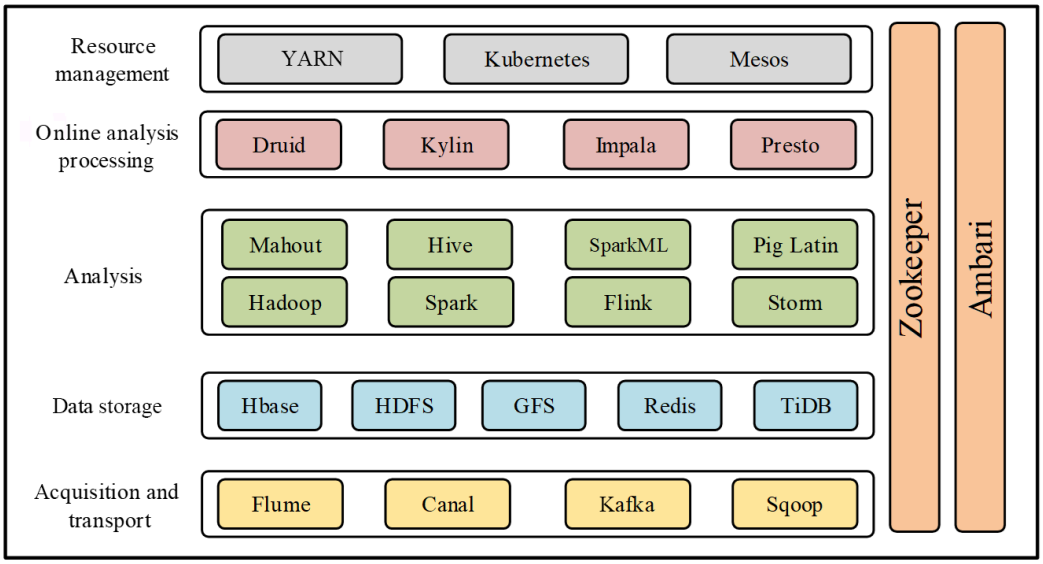

Fig. 1 Distributed computing technology stack for handling big data.

Therefore, executing a complex iterative algorithm over a big data set on a cluster efficiently is a challenge. A distributed computing framework is essential to support the execution.

3.1 Basic steps of distributed computing

On the basis of the divide-and-conquer strategy, a big data set to be analyzed is partitioned into small data block files, which are stored in a distributed manner on the nodes of a cluster and managed in a distributed file system, e.g., HDFS. The following two basic steps of distributed computing are used to compute the big data set on a cluster with the shared-nothing architecture[86, 87].

(1) Local operation. On each node, the local data files are read into the memory of local nodes and computed by the local operations to produce the local results. The local operations are executed independently on each node and in parallel among the nodes of the cluster. This step is called the Map operation in MapReduce.

(2) Global operation. The local results are transferred to some nodes, e.g., the master node, on which the global results are computed from the local results by the global operations. This step is called the Reduce operation in MapReduce.

Figure 2 illustrates the two basic steps of distributed computing. The data block files are stored on the disks of N nodes. The local operation is assigned to each node to operate on the local data blocks. On each node, it reads the local data block file into the memory and operates on it to generate the local result, which may be written back to the local disk, e.g., Hadoop MapReduce. The global operation takes the local result from each node and transfers it to the master node to generate the global result. The total computation cost of the two steps can be divided into three parts.

(1) Cost of reading and writing local files on each node called I/O costs;

(2) Cost of transferring local results to the nodes for calculating the global results called communication costs;

(3) Cost of executing the local and global operations called computing costs.

All costs occur once if a computing task is completed by a distributed algorithm in one iteration on the dataset, e.g., the computing task of counting the word frequencies of web pages. However, if a distributed dataset is analyzed by an iterative algorithm, then the above costs will repeatedly occur with respect to the number of iterations to complete the computation. Therefore, an important task is to minimize these costs in designing distributed computing frameworks and algorithms for big data analysis. In the following subsections, we review how the above costs in the two basic steps of distributed computing are handled in Hadoop MapReduce, Haloop and Spark systems.

3.2 Hadoop MapReduce

Hadoop is a quasi-standard platform for distributed big data applications[88]. In the Hadoop MapReduce computing framework, an iterative algorithm is decomposed into a sequence of pairs of mapper and reducer operations, that are executed sequentially in a pipeline manner[89]. One pair of mapper and reducer operations is taken as an example.

(1) In the mapper phase, the data block files stored in HDFS are read into the memory of local nodes separately. The predefined mapper operation is performed on each

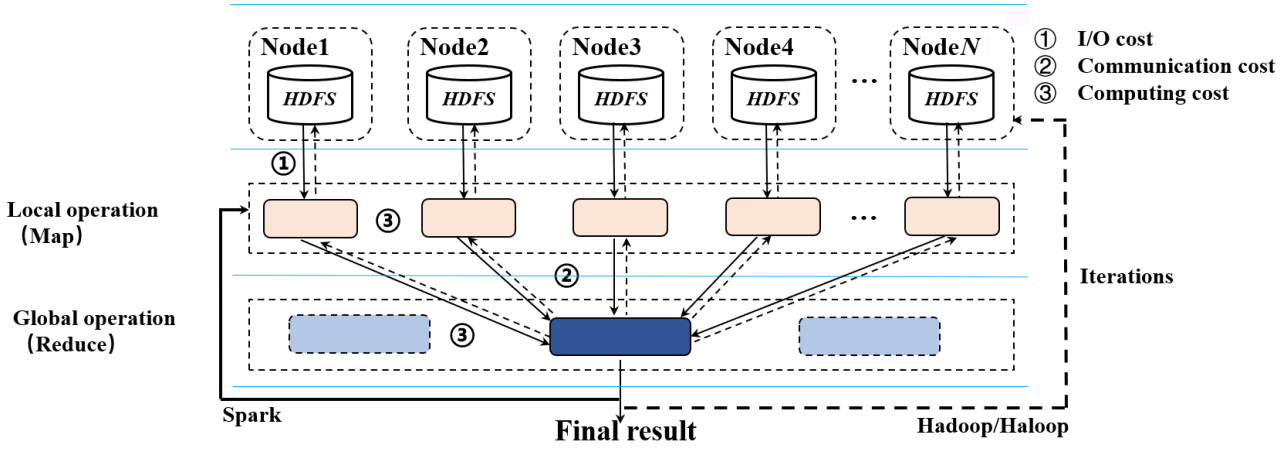

Fig. 2 MapReduce-type distributed computing framework. Hadoop and Haloop iterative algorithms generate I/O, communication, and computing costs repeatedly, whereas Spark reduces I/O costs significantly by using in-memory computing.

local node independently and in parallel. When the operation is complete, the local result is written to the local disk. The local result is read back into the memory in the reducer operation that follows.

(2) Before the reducer operation, the internal shuffle operation fetches all local results from the local nodes, shuffles the local data on the reduce keys, and temporarily saves the corresponding data blocks to the nodes[90] where the reducer operations will be performed. Then, the reducer operation reads the shuffled local result data to the memory of the reduce node and operates on the data to generate the global results as the intermediate results for the next pair of mapper and reducer operations if the current pair is not the last one or as the final results of the algorithm if the pair is the last one. In this step, the shuffle operation requires the local results to be transferred to corresponding reduce nodes, which increases communication costs.

Evidently, expensive communication costs occur from the mapper operation to the reducer operation in each pair. If the sequence of mapper and reducer operations contains a large number of pairs, then the shuffle operation will repeatedly occur and generate high communication costs. Unfortunately, this communication cost cannot be avoided in the MapReduce computing framework.

When the iterative algorithms are executed, the I/O cost is also high in Hadoop MapReduce because the I/O operations occur in each mapper and reducer operation. Both mapper and reducer operations are performed in {key, value} data format; thus, implementing an iterative algorithm in this style is not always efficient. This implementation style affects the execution efficiency of some algorithms.

3.3 Haloop

Haloop is a Hadoop variant that retains the same computing mechanism as Hadoop[91]. It was designed to improve the computing efficiency of Hadoop on iterative tasks with the following modifications.

- A loop control mechanism is used in Haloop to control the number of iterations by grouping several pairs of mapper and reducer operations in a subtask to reduce the number of subtasks, each group being carried out in one iteration.
- The invariant data multiplexed in the iterative calculation is cached and indexed to local disks to avoid unnecessary network I/O operations[92, 93].
- To reduce the communication cost, the task is scheduled to run on the local data as much as possible[94].

With the above modifications on Hadoop, Haloop only partially reduces the I/O, communication and computing costs, and increases the overall computing performance over Hadoop. However, these three types of costs are still high for complex iterative tasks over big data sets because they are not completely removed from the computing framework. Therefore, Haloop is not widely used in real applications.

3.4 Spark

Spark is a distributed computing system for big data analysis that was designed and implemented from scratch. It adopts the in-memory computing technology to improve the computing performance of big data analysis. To reduce I/O costs, Spark uses a unique read-only data structure called RDD to hold a distributed data file in memory[95] to avoid repeated I/O operations.

An RDD consists of a set of partitions, and each partition holds one data block of a distributed data set on a local node. The set of partitions is distributed among the nodes of a cluster. An RDD is used to hold the input dataset or an intermediate result. The transformation operators are defined to operate on existing RDDs and the results are saved in a new RDD. A data analysis task is implemented as a sequence of transformation operators on a set of RDDs, and the sequence is represented as a DAG that can be decomposed into stages that represent the topological sorting of the overall computing[96].

Each stage in a DAG diagram is composed of a set of RDDs for interactive computing, which are encapsulated into tasksets in a one-to-one correspondence with TasksetManager. Spark DAGScheduler schedules the execution of DAG to complete distributed computing as follows.

(1) The local data block files are read into the partitions of input RDD on the local nodes. TaskScheduler selects a taskset without pre-dependency from the DAG and distributes tasks to each computing node. Each node completes the calculation locally and in parallel with other nodes.

(2) The local partition on each node is aggregated through the shuffle operation, and the results are saved in the memory for use in the following stages. When the execution reaches the last stage which is an action operator, the global result is written to HDFS.

A Spark data analysis application is performed through a sequence of operators on a set of RDDs.

Some operators operate on local partitions independently without considering other partitions. Some operators operate on partitions on different nodes, requiring data join or shuffle operations, which result in increased data communication costs. Therefore, the spark system for big data analysis cannot avoid communication costs but can avoid many I/O operations and improve computing efficiency.

Compared with the operations on local partitions that run independently, the join and shuffle operations are still not efficient due to data transfer between nodes. Therefore, Spark-based big data analysis still has high communication costs. Another bottleneck in Spark-based big data analysis is that the input data cannot exceed the memory capacity. Otherwise, Spark will become slow in carrying out such applications or will be unable to do so if the input dataset is too big. As such, Spark is still not scalable to big data sets, e.g., in terabytes. Another problem is that the iterative algorithms still need to be rewritten in MapReduce for distributed computing, thus hampering the use of many good serial algorithms in big data analysis.

3.5 Discussions

To improve the performance of distributed computing for big data analysis, the I/O, communication, and computation costs must be minimized. One solution is to enhance the hardware components, such as by using high-speed solid state drives (SSDs) on the nodes to speed up the I/O operations, using the high-speed cluster switch to speed up the data transfer among the nodes, and the using a high-speed CPU and large memory to speed up the local and global computation. However, this solution will increase financial costs because expensive hardware needs to be purchased.

Other solutions enhance the computing frameworks to reduce the total cost of distributed computing for big data analysis. For example, Haloop enhanced Hadoop MapReduce by reducing the I/O and communication costs. Spark took a revolutionary step to define the RDD to store data in the memory to support in-memory computing, thus increasing the computing efficiency of data analysis algorithms. However, the communication costs still occur because of its MapReduce style of distributed computing.

Another issue is that the above-mentioned computing frameworks require the computation of an entire dataset because missing any data blocks of a big data set can result in inaccurate or incorrect results. Therefore, they are not scalable to extremely big data sets in terabytes or greater. In the next section, we review a new non-MapReduce distributed computing framework for big data analysis, which is scalable to very big data sets and supports approximate big data computing.

4 Non-MapReduce Distributed Computing for Big Data Analysis

The distributed computing frameworks discussed in the previous section have two issues that still affect the computing performance of big data analysis: high costs of data communication among the nodes and scalability to big data sets that are much larger than the memory size. The first issue is caused by the MapReduce programming model of implementing iterative algorithms on the distributed computing frameworks. In this model, an iterative algorithm has to be rewritten as a sequence of Map and Reduce operations and executed in steps under the distributed computing framework. The algorithm also requires taking the whole dataset to compute the final result, which affects the scalability due to the memory limit.

In this section, we review a new approach of distributed computing for big data analysis to solve the above two problems.

4.1 RSP data model

To reduce the costs of data communication among the nodes, a new data representation method called the RSP data model was proposed in Refs. [12, 97] to enable the computation for distributed big data analysis in one pair of local and global operations, even with an iterative algorithm. In the RSP data model, a big data file is partitioned into a set of disjoint RSP data blocks[13], which can be stored as an HDFS file. The difference between this new data representation from an ordinary HDFS file is that the collective frequency distributions of RSP data blocks are similar to each other and to the collective frequency distributions of the entire big data file[14, 98, 99]. As such, the RSP data blocks can be used as random samples of a big data file to estimate the statistical results of the big data file.

Representing a big data file as a set of RSP data blocks enables a new distributed big data analysis approach, in which an approximate result of a big data set can be computed from a series of randomly selected RSP data blocks. The data block is analyzed by a serial algorithm running on a node to produce the approximate result. To improve the result accuracy, multiple RSP data

blocks can be selected randomly and analyzed by the same algorithm on different nodes independently and in parallel to produce multiple approximate results. These results are transferred to the master node and integrated into the final result through the global operation.

With the use of the new approach in big data analysis, an important step is to convert the data blocks of a big HDFS file to a set of RSP data blocks, which are also saved as an RSP-HDFS file. An algorithm was proposed to complete this conversion in two stages[100]. In the first stage, each HDFS data block is randomized using a shuffle operation, and the intrablock shuffle operation is not involved in any mutual data communication between nodes. In the second stage, the shuffled data block is partitioned to a number of subblocks equal to the number of the RSP data blocks to be converted. Then, the same number of containers is created, and one subblock is randomly selected without replacement from each shuffled data block and stored in the container. The same operation is performed on all shuffled HDFS data blocks in parallel. Finally, after all containers are filled with data results, they are saved as an RSP-HDFS file.

The conversion from an HDFS file to an RSP data model is a one-off operation that can be performed offline[101, 102]. The RSP data model also enables block-level sampling, which is much faster than record-level sampling, which is time consuming on big distributed data files. The RSP data blocks are persistent as the local data files; thus, if an RSP block file is selected as a random sample, then it is read into memory in seconds. Therefore, taking multiple random samples from a big distributed data file is no longer a time-consuming task. At the same time, the diversity of sample data blocks and the consistency of feature distributions are ensured, thereby enabling approximate computing on large-scale data and reducing the computing overhead of big data analysis significantly.

4.2 Non-MapReduce distributed computing framework

A non-MapReduce distributed computing framework enabled by the RSP data model is illustrated in Fig. 3. This computing framework also supports the two basic operations of distributed computing: local operation and global operation. However, unlike in Fig. 3, the local and global operations in this framework are separately conducted, i.e., the local operation first and then the global operation. When an iterative algorithm is involved in the local operation, it completes all iterations in the memory of its local node to generate the local result before the global operation starts. What this process entails is only one data communication cost, i.e., transferring the local results to the nodes of global operation. As such, the data communication costs are significantly reduced in this non-MapReduce distributed computing framework.

Another important feature of this computing framework is that the data analysis algorithm is run independently to analyze an RSP data block on a local node, and it does not need to be rewritten as a sequence of Map and Reduce operations. Any serial algorithm is applicable in this fashion, and the extensive effort to reimplement the serial algorithms for parallel executions in distributed computing can be saved. Serial algorithms that cannot be parallelized effectively can be directly

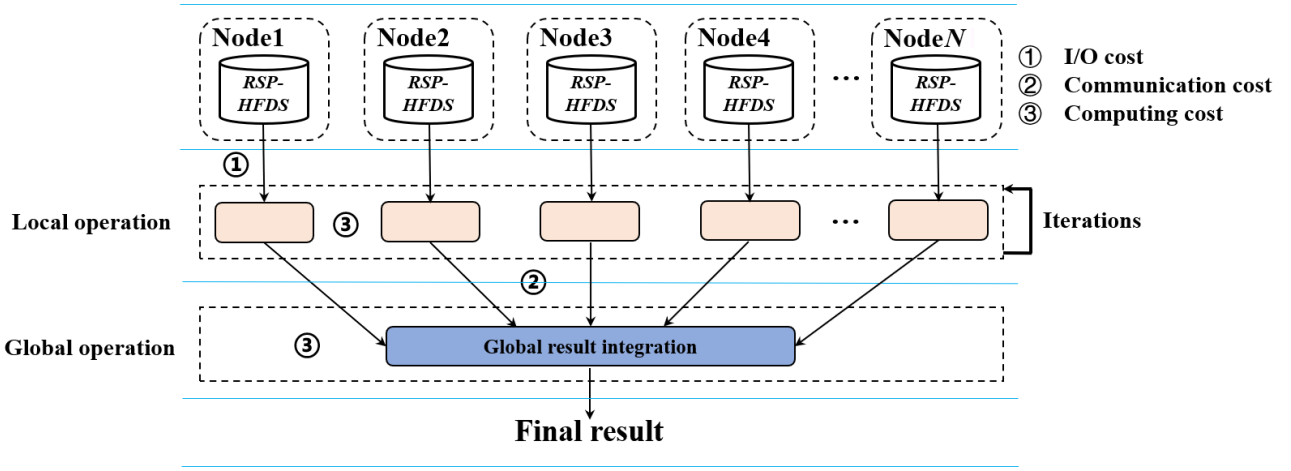

Fig. 3 Non-MapReduce distributed computing framework. Enabled by the RSP data model, this framework reduces both I/O and communication costs significantly when running iterative algorithms and supports approximate computing, thus being efficient and scalable to big data.

used in this framework for distributed computing.

Unlike the MapReduce distributed computing framework, which requires dividing both the big data set into data blocks and the algorithm into pairs of mapper and reducer operations, the non-MapReduce framework requires only the division of the big data set into RSP data blocks. The RSP data blocks are the random samples of big data sets; thus, the algorithm can analyze them independently and in parallel to produce unbiased estimate results. To improve the accuracy of big data analysis, the local results are integrated into the final result with the global operation. Not only is the final result better than the local results according to the central limit theorem, but the confidence interval of the final result can also be computed with a given significance level. Therefore, the final result computed with the non-MapReduce computing framework is an estimation of big data statistics, not an exact computed result. It computes only a few randomly selected RSP data blocks; therefore, it holds the whole dataset in memory, like Spark does, thus removing the memory bottleneck. In this sense, it is scalable to extremely big data sets whenever their RSP data models are available.

4.3 Approximate computing for big data analysis

Approximate computing can be defined as a computing strategy that uses a part of the entire big data set to compute an approximate result as an estimate of the result that is computed from the entire big data set. In statistical analysis, approximate computing is equivalent to using a random sample to compute an estimate of the population. The approximate result is required to be unbiased and accurate. Taking random samples from a big distributed file is time consuming[103–109], which is why the traditional MapReduce distributed computing frameworks do not support approximate computing efficiently.

The non-MapReduce distributed computing framework was designed to support approximate computing for big data analysis[110]. The RSP data model provides ready-to-use random samples for approximate computing. Instead of record-level sampling that needs to go through the entire set of records, block-level sampling is used to randomly select RSP blocks and read the RSP blocks into the memory of local nodes for the algorithm to analyze. This approach essentially solves the problem of sampling a big distributed data file on a cluster[97, 111, 112]. The sampling time is reduced from minutes or even hours to seconds when an extremely big data file is being sampled.

The non-MapReduce distributed computing framework supports approximate computing of multiple random samples because multiple RSP data blocks are used in the local operation to compute the multiple approximate results. Then, the multiple approximate results are integrated into the ensemble result which is statistically better than any individual local approximate result. In the meantime, the distribution of the statistic or the random function computed from the random sample can be estimated from the set of local approximate results, and the confidence interval of the ensemble result can also be calculated. The estimation quality can be improved by increasing the number of selected RSP data blocks, and more RSP data blocks do not affect the computing efficiency considerably. As such, approximate computing has statistical advantages over the exact computing of the whole dataset. It is scalable to big data sets and produces ensemble models that are more accurate than the single model from the whole dataset.

4.4 Discussion

The non-MapReduce distributed computing framework provides the following advantages to support big data analysis.

(1) It reduces the data communication costs significantly when an iterative algorithm is executed because the RSP data representation allows each RSP data block to be analyzed as a random sample of the big data file by the algorithm locally and independently. No data communication is required when multiple RSP data blocks are computed in parallel on different nodes.

(2) It supports the running of serial algorithms over local RSP data blocks on multiple nodes independently without the needs of rewriting them in MapReduce. Therefore, more analytical algorithms can be used in distributed big data analysis.

(3) It supports approximate computing so that the memory is no longer a bottleneck because the entire dataset does not need to be read into memory. Therefore, it is scalable to ultra-large-scale data in terabytes.

Aside from having the above advantages, the new computing framework also ensures excellent fault tolerance because the final result can be computed from the subset of the approximate local results if some local nodes fail. It also improves the efficiency of big data exploration and cleaning[113].

5 Evaluation

In this section, we use nine metrics to evaluate three distributed computing frameworks with respect to big data analysis, together with two systems that are widely used in data analysis, namely, Python and R. Haloop is not included because it is a variant of Hadoop and is not widely used. The results are given in Table 1. The first three metrics are I/O costs, communication costs, and computing costs, which were explained before. The other six metrics are defined as follows.

• **Computing efficiency**. Given the same computing resources and the same dataset, this metric measures the completion time of the same algorithm that runs on the same big data set in a different computing framework. A short completion time corresponds to high computing efficiency.

• **Computing style**. We have two styles of computing, e.g., exact computing, which uses the entire dataset to compute an exact result and approximate computing, which uses a small subset of random data blocks to compute an approximate result. Given that only a small and representative portion of big data is computed, approximate computing is much more efficient than exact computing in big data analysis.

• **Scalability**. This metric measures whether a computing framework can scale as the size of input datasets increases, the data analysis algorithms still run, and the final result can be produced on a fixed hardware resource.

• **Algorithm serialization**. This metric measures whether the serial algorithms can be used in a computing framework directly without the need to rewrite in MapReduce style. We can see that only the non-MapReduce framework in distributed computing allows the direct execution of serial algorithms.

• **Data throughput**. This metric measures how big a dataset can be computed in a certain time.

• **Iteration support**. This metric indicates whether a framework can support the efficient execution of iterative data analysis algorithms.

From Table 1, we can see that Python and R are not suitable for big data analysis because they run on limited computing resources. Data scientists usually use these systems to analyze samples of big data sets. However, for complex big data, a sample itself is a big data set in its own right and requires heavy computing resources for analysis.

For the three distributed computing frameworks, Hadoop MapReduce is not suitable for big data analysis by using complex iterative algorithms because of its low computing efficiency, low scalability, and data throughput. Spark has a high computing efficiency if the big data set can be held in memory. However, if the size of the big data set is larger than the memory of the cluster system, then its computing efficiency will become low. Therefore, Spark is not scalable to the data when the size of big data exceeds the size of the system memory. Both Hadoop and Spark cannot use serial algorithms directly in distributed computing. Comparatively, the non-MapReduce distributed computing framework has clear advantages in big data analysis. It is efficient and scalable to big data, has high data throughput, and can directly use serial algorithms in distributed computing. Although it produces approximate results from multiple samples of a big data set, statistically, the error of approximation can be controlled in an acceptable confidence interval by increasing the size and number of RSP data blocks. Therefore, the non-MapReduce computing framework has the potential to become the main distributed computing framework to support big data analysis.

6 Conclusion

In this paper, we reviewed the distributed computing frameworks that were designed for handling big data in

Table 1 Evaluations of computing frameworks for big data analysis.

	Hadoop	Spark	Non-MapReduce	Python and R
I/O costs	High	Low	Low	Low
Communication costs	High	High	Low	No
Computing costs	High	Middle	Low	High
Computing efficiency	Low	High	Very high	Low
Computing style	Exact	Exact	Approximate	Exact
Scalability	No	No	Yes	No
Algorithm serialization	No	No	Yes	Yes
Data throughput	Low	High (in-memory)	High	Low
Iteration support	Low	High (in-memory)	High	Low

general and supporting big data analysis in particular. These frameworks were designed for clusters of shared-nothing architecture, which supports distributed computing based on the divide-and-conquer strategy. The classical MapReduce type of distributed computing, which divides both data and algorithms, processes big data efficiently using non-iterative algorithms. However, it is not efficient for big data analysis when the complex iterative algorithms are used due to the data communication costs and in-memory computing of the whole dataset.

The non-MapReduce distributed computing framework runs serial algorithms on local data blocks independently and in parallel to generate local results and then integrate the local results into the final result. This strategy only divides the big data set but does not parallelize the serial algorithm to produce the local results. Therefore, the data communication costs only occur in the integration of local results, thus reducing the data communication costs significantly. Other two important advantages are the direct use of serial algorithms in generating local results and the approximate computing ability, which uses only a subset of randomly selected data blocks to compute an approximate result in a range of user-controlled accuracy. These two advantages are enabled by the RSP data model. The non-MapReduce distributed computing framework can be expected to appear in the new generation of big data computing technology and platforms because it is efficient in big data analysis and scalable to extremely big data in terabytes and greater. Therefore, non-MapReduce distributed computing and related topics are likely to become interesting new research directions in big data computing.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 61972261) and Basic Research Foundations of Shenzhen (Nos. JCYJ 20210324093609026 and JCYJ20200813091134001).

References

[1] M. Anjomshoa, M. Salleh, and M. P. Kermani, A taxonomy and survey of distributed computing systems, *J. Appl. Sci.*, vol. 15, no. 1, pp. 46–57, 2015.

[2] D. C. Marinescu, Parallel and distributed computing: Memories of time past and a glimpse at the future, in *Proc. 2014 IEEE 13th Int. Symp. Parallel and Distributed Computing*, Marseille, France, 2014, pp. 14&15.

[3] J. Fan, F. Han, and H. Liu, Challenges of big data analysis, *Natl. Sci. Rev.*, vol. 1, no. 2, pp. 293–314, 2014.

[4] Z. N. Rashid, S. R. M. Zebari, K. H. Sharif, and K. Jacksi, Distributed cloud computing and distributed parallel computing: A review, in *Proc. 2018 Int. Conf. Advanced Science and Engineering* (*ICOASE*), Duhok, Iraq, 2018, pp. 167–172.

[5] V. K. Singh, M. Taram, V. Agrawal, and B. S. Baghel, A literature review on hadoop ecosystem and various techniques of big data optimization, in *Advances in Data and Information Sciences*, M. Kolhe, M. Trivedi, S. Tiwari, and V. Singh, eds. Singapore: Springer, 2018, pp. 231–240.

[6] K. Zhang, B. Qin, and Q. C. Liu, Study of parallel computing framework based on GPU-Hadoop, (in Chinese), *Applicat. Res. Comput.*, vol. 31, no. 8, pp. 2548–2550& 2556, 2014.

[7] H. Ogawa, H. Nakada, R. Takano, and T. Kudoh, SSS: An implementation of key-value store based MapReduce framework, in *Proc. 2010 IEEE Second Int. Conf. Cloud Computing Technology and Science*, Indianapolis, IN, USA, 2010, pp. 754–761.

[8] S. Ghemawat, H. Gobioff, and S. T. Leung, The google file system, *ACM SIGOPS Oper. Syst. Rev.*, vol. 73, no. 5, pp. 29–43, 2003.

[9] V. K. C. Bumgardner, V. W. Marek, and C. D. Hickey, Cresco: A distributed agent-based edge computing framework, in *Proc. 2016 12th Int. Conf. Network and Service Management* (*CNSM*), Montreal, Canada, 2016, pp. 400–405.

[10] M. Grivas and D. Kehagias, A multi-platform framework for distributed computing, in *Proc. 2008 Panhellenic Conf. Informatics*, Samos, Greece, 2008, pp. 163–167.

[11] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, The hadoop distributed file system, in *Proc. 2010 IEEE 26th Symp. Mass Storage Systems and Technologies* (*MSST*), Incline Village, NV, USA, 2010, pp. 1–10.

[12] S. Salloum, J. Z. Huang, Y. He, and X. Chen, An asymptotic ensemble learning framework for big data analysis, *IEEE Access*, vol. 7, pp. 3675–3693, 2018.

[13] J. Z. Huang, Y. He, C. Wei, and X. Zhang, Random sample partition data model and related technologies for big data analysis, *J Data Acquisit. Process.*, vol. 34, no. 3, pp. 373–385, 2019.

[14] Y. He, J. Z. Huang, H. Long, Q. Wang, and C. Wei, I-sampling: A new block-based sampling method for large-scale dataset, in *Proc. 2017 IEEE Int. Congress on Big Data* (*BigData Congress*), Honolulu, HI, USA, 2017, pp. 360–367.

[15] T. Biswas, P. Kuila, and A. K. Ray, Multi-level queue for task scheduling in heterogeneous distributed computing system, in *Proc. 2017 4th Int. Conf. Advanced Computing and Communication Systems* (*ICACCS*), Coimbatore, India, 2017, pp. 1–6.

[16] L. Globa and N. Gvozdetska, Comprehensive energy efficient approach to workload processing in distributed computing environment, in *Proc. 2020 IEEE Int. Black Sea Conf. Communications and Networking* (*BlackSeaCom*), Odessa, Ukraine, 2020, pp. 1–6.

[17] N. A. Bahnasawy, M. A. Koutb, M. Mosa, and F. Omara, A new algorithm for static task scheduling for heterogeneous distributed computing systems, *Afr. J. Mathemat. Comput.*

*S*ci. Res., vol. 4, no. 6, pp. 221–234, 2011.

[18] M. I. Daoud and N. Kharma, A high performance algorithm for static task scheduling in heterogeneous distributed computing systems, *J. Parallel Distrib. Comput.*, vol. 68, no. 4, pp. 399–409, 2008.

[19] H. Lin, X. Zhu, B. Yu, X. Tang, W. Xue, W. Chen, L. Zhang, T. Hoefler, X. Ma, X. Liu, et al., ShenTu: Processing multi-trillion edge graphs on millions of cores in seconds, in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, Dallas, TX, USA, 2018, pp. 706–716.

[20] X. J. Yang, X. K. Liao, K. Lu, Q. F. Hu, J. Q. Song, and J. S. Su, The TianHe-1A supercomputer: Its hardware and software, *J. Comput. Sci. Technol.*, vol. 26, no. 3, pp. 344–351, 2011.

[21] D. P. Anderson, E. Korpela, and R. Walton, High-performance task distribution for volunteer computing, in *Proc. First Int. Conf. e-Science and Grid Computing* (*e-Science'05*), Melbourne, Australia, 2005, p. 8.

[22] G. Lu and W. H. Zeng, Cloud computing survey, *Appl. Mechan. Mater.*, vols. 530-531, pp. 650–661, 2014.

[23] K. Krauter, R. Buyya, and M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, *Software Pract. Exper*, vol. 32, no. 2, pp. 135–164, 2002.

[24] S. Patidar, D. Rane, and P. Jain, A survey paper on cloud computing, in *Proc. 2012 Second Int. Conf. Advanced Computing & Communication Technologies*, Rohtak, India, 2012, pp. 394–398.

[25] R. Nath and A. Nagaraju, A novel task assignment heuristic using local search in distributed computing systems, in *Proc. 2017 Int. Conf. Energy, Communication, Data Analytics and Soft Computing* (*ICECDS*), Chennai, India, 2017, pp. 2767–2771.

[26] Z. Fadika and M. Govindaraju, DELMA: Dynamically elastic MapReduce framework for CPU-intensive applications, in *Proc. 2011 11^{th} IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing*, Newport Beach, CA, USA, 2011, pp. 454–463.

[27] K. Singh, M. Alam, and S. Kumar, A survey of static scheduling algorithm for distributed computing system, *Int. J. Comput. Applicat.*, vol. 129, no. 2, pp. 25–30, 2015.

[28] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, Diagnosing performance variations in HPC applications using machine learning, in *Proc. 32^{nd} High Performance Computing*, Frankfurt, Germany, 2017, pp. 355–373.

[29] G. Ramirez-Gargallo, M. Garcia-Gasulla, and F. Mantovani, Tensorflow on state-of-the-art HPC clusters: A machine learning use case, in *Proc. 2019 19^{th} IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing* (*CCGRID*), Larnaca, Cyprus, 2019, pp. 526–533.

[30] V. K. Naik, S. K. Setia, and M. S. Squillante, Performance analysis of job scheduling policies in parallel supercomputing environments, in *Proc. 1993 ACM/IEEE Conf. Supercomputing*, Portland, OR, USA, 1993, pp. 824–833.

[31] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, HPC cloud for scientific and business applications: taxonomy, vision, and research challenges, *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–29, 2019.

[32] T. D. Thanh, S. Mohan, E. Choi, S. Kim, and P. Kim, A taxonomy and survey on distributed file systems, in *Proc. 2008 4^{th} Int. Conf. Networked Computing and Advanced Information Management*, Gyeongju, Republic of Korea, 2008, pp. 144–149.

[33] J. Blomer, A survey on distributed file system technology, *J. Phys. Conf. Ser.*, vol. 608, p. 012039, 2015.

[34] L. Jiang, B. Li, and M. Song, The optimization of HDFS based on small files, in *Proc. 2010 3^{rd} IEEE Int. Conf. Broadband Network and Multimedia Technology* (*IC-BNMT*), Beijing, China, 2010, pp. 912–915.

[35] S. Zhuo, X. Wu, W. Zhang, and W. Dou, Distributed file system and classification for small images, in *Proc. 2013 IEEE Int. Conf. Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, Beijing, China, 2013, pp. 2231–2234.

[36] S. Fu, L. He, C. Huang, X. Liao, and K. Li, Performance optimization for managing massive numbers of small files in distributed file systems, *IEEE Trans. Parallel Distribut. Syst.*, vol. 26, no. 12, pp. 3433–3448, 2015.

[37] H. Che and H. Zhang, Exploiting fastDFS client-based small file merging, in *Proc. 2016 Int. Conf. Artificial Intelligence and Engineering Applications*, Hong Kong, China, 2016, pp. 242–246.

[38] W. K. Josephson, L. A. Bongo, K. Li, and D. Flynn, DFS: A file system for virtualized flash storage, *ACM Trans. Stor.*, vol. 6, no. 3, p. 14, 2010.

[39] Z. Ullah, S. Jabbar, M. H. Bin Tariq Alvi, and A. Ahmad, Analytical study on performance, challenges and future considerations of Google file system, *Int. J. Computer Communicat. Eng.*, vol. 3, no. 4, pp. 279–284, 2014.

[40] R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, and Y. Huang, SHadoop: Improving MapReduce performance by optimizing job execution mechanism in hadoop clusters, *J. Parallel Distribut. Comput.*, vol. 74, no. 3, pp. 2166–2179, 2014.

[41] I. Polato, R. Ré, A. Goldman, and F. Kon, A comprehensive view of hadoop research—A systematic literature review, *J. Network Comput. Applicat.*, vol. 46, pp. 1–25, 2014.

[42] Y. Wang, W. Jiang, and G. Agrawal, SciMATE: A novel MapReduce-like framework for multiple scientific data formats, in *Proc. 2012 12^{th} IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing* (*CCGRID 2012*), Ottawa, Canada, 2012, pp. 443–450.

[43] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, *Commun ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[44] M. R. Ghazi and D. Gangodkar, Hadoop, MapReduce and HDFS: A developers perspective, *Proc. Comput. Sci.*, vol. 48, pp. 45–50, 2015.

[45] Y. Zhang, Q. Gao, L. Gao, and C. Wang, iMapReduce: A distributed computing framework for iterative computation, *J. Grid Comput.*, vol. 10, no. 1, pp. 47–68, 2012.

[46] H. Alshammari, J. Lee, and H. Bajwa, H2Hadoop: Improving hadoop performance using the metadata of related jobs, *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 1031–1040, 2018.

[47] P. S. Janardhanan and P. Samuel, Launch overheads of spark applications on standalone and hadoop YARN clusters, in *Advances in Electrical and Computer Technologies*, T. Sengodan, M. Murugappan, and S. Misra, eds. Singapore: Springer, 2020, pp. 47–54.

[48] C. Lam, *Hadoop in Action*. Lewis Street Greenwich, CT, USA: Manning Publications Co., 2010.

[49] S. R. Alapati, *Expert Hadoop Administration: Managing, Tuning, and Securing Spark, YARN, and HDFS*. Palo Alto, CA, USA: Addison-Wesley Professional, 2016.

[50] R. K. Bhatia and A. Bansal, Deploying and improving hadoop on pseudo-distributed mode, *Compusoft*, vol. 3, no. 10, p. 1136, 2014.

[51] F. Li, J. Chen, and Z. Wang, Wireless MapReduce distributed computing, *IEEE Trans. Inform. Theory*, vol. 65, no. 10, pp. 6101–6114, 2019.

[52] C. F. Chiu, S. J. Hsu, and S. R. Jan, Distributed MapReduce framework using distributed hash table, in *Proc. 2013 Int. Joint Conf. Awareness Science and Technology & Ubi-Media Computing* (*iCAST 2013 & UMEDIA 2013*), Aizu-Wakamatsu, Japan, 2013, pp. 475–481.

[53] S. D. Kavila, G. S. V. P. Raju, S. C. Satapathy, A. Machiraju, G. V. L. Kinnera, and K. Rasly, A survey on fault management techniques in distributed computing, in *Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications* (*FICTA*), S. C. Satapathy, S. K. Udgata, and B. N. Biswal, eds. Berlin, Germany: Springer, 2013, pp. 593–602.

[54] Y. C. Sun and X. F. Wang, MapReduce designed to optimize computing model based on hadoop framework, (in Chinese), *Comput. Sci.*, vol. 41, no. 11A, pp. 333–336, 2014.

[55] J. Yu, J. Wu, and M. Sarwat, A demonstration of geoSpark: A cluster computing framework for processing big spatial data, in *Proc. 2016 IEEE 32nd Int. Conf. Data Engineering* (*ICDE*), Helsinki, Finland, 2016, pp. 1410–1413.

[56] Z. Yang, C. Zhang, M. Hu, and F. Lin, OPC: A distributed computing and memory computing-based effective solution of big data, in *Proc. 2015 IEEE Int. Conf. Smart City/ SocialCom/SustainCom* (*SmartCity*), Chengdu, China, 2015, pp. 50–53.

[57] V. Taran, O. Alienin, S. Stirenko, Y. Gordienko, and A. Rojbi, Performance evaluation of distributed computing environments with Hadoop and spark frameworks, in *Proc. 2017 IEEE Int. Young Scientists Forum on Applied Physics and Engineering* (*YSF*), Lviv, Ukraine, 2017, pp. 80–83.

[58] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al., Spark SQL: Relational data processing in spark, in *Proc. 2015 ACM SIGMOD Int. Conf. Management of Data*, Melbourne, Australia, 2015, pp. 1383–1394.

[59] Y. Benlachmi and M. L. Hasnaoui, Big data and spark: Comparison with hadoop, in *Proc. 2020 Fourth World Conf. Smart Trends in Systems, Security and Sustainability* (*WorldS4*), London, UK, 2020, pp. 811–817.

[60] P. Karunaratne, S. Karunasekera, and A. Harwood, Distributed stream clustering using micro-clusters on apache storm, *J. Parallel Distribut. Comput.*, vol. 108, pp. 74–84, 2017.

[61] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas, State management in Apache Flink®: Consistent stateful distributed stream processing, *Proc. VLDB Endowm.*, vol. 10, no. 12, pp. 1718–1729, 2017.

[62] F. Hueske and V. Kalavri, *Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[63] 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. Technol. Committ. Data Eng.*, vol. 36, no. 4, pp. 28–38, 2015.

[64] A. Katsifodimos and S. Schelter, Apache Flink: Stream analytics at scale, in *Proc. 2016 IEEE Int. Conf. Cloud Engineering Workshop* (*IC2EW*), Berlin, Germany, 2016, p. 193.

[65] M. H. Iqbal and T. R. Soomro, Big data analysis: Apache storm perspective, *Int. J. Computer Trends Technol.*, vol. 19, no. 1, pp. 9–14, 2015.

[66] T. Da Silva Morais, Survey on frameworks for distributed computing: Hadoop, spark and storm, in *Proc. 10th Doctoral Symp. in Informatics Engineering–DSIE'15*, Porto, Portugal, 2015, pp. 95–105.

[67] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, Hive: A warehousing solution over a map-reduce framework, *Proc. VLDB Endowm.*, vol. 2, no. 2, pp. 1626–1629, 2009.

[68] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, Building a high-level dataflow system on top of map-reduce: The pig experience, *Proc. VLDB Endowm.*, vol. 2, no. 2, pp. 1414–1425, 2009.

[69] A. Thusoo, J. Sen Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, Hive-a petabyte scale data warehouse using hadoop, in *Proc. 2010 IEEE 26th Int. Conf. Data Engineering* (*ICDE 2010*), Long Beach, CA, USA, 2010, pp. 996–1005.

[70] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, Pig latin: A not-so-foreign language for data processing, in *Proc. 2008 ACM SIGMOD Int. Conf. Management of data*, Vancouver, Canada, 2008, pp. 1099–1110.

[71] X. Lu, H. Shi, R. Biswas, M. H. Javed, and D. K. Panda, DLoBD: A comprehensive study of deep learning over big data stacks on HPC clusters, *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 4, pp. 635–648, 2018.

[72] S. Sakr, Big data processing stacks, *IT Profess.*, vol. 19, no. 1, pp. 34–41, 2017.

[73] J. Kreps, N. Narkhede, and J. Rao, Kafka: A distributed messaging system for log processing, in *Proc. 6th Int. Workshop on Networking Meets Databases*, Athens, Greece, 2011, pp. 1–7.

[74] S. Aravinth, A. H. Begam, S. Shanmugapriyaa, S. Sowmya, and E. Arun, An efficient HADOOP frameworks SQOOP and ambari for big data processing, *Int. J. Innovat. Res. Sci. Technol.*, vol. 1, no. 10, pp. 252–255, 2015.

[75] M. N. Vora, Hadoop-Hbase for large-scale data, in *Proc.*

2011 Int. Conf. Computer Science and Network Technology, Harbin, China, 2011, pp. 601–605.
[76] J. Carlson, *Redis in action*. Shelter Island, NY, USA: Manning, 2013.
[77] D. Huang, Q. Liu, Q. Cui, Z. Fang, X. Ma, F. Xu, L. Shen, L. Tang, Y. Zhou, M. Huang, et al., TiDB: A raft-based HTAP database, *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3072–3084, 2020.
[78] R. Anil, G. Capan, I. Drost-Fromm, T. Dunning, E. Friedman, T. Grant, S. Quinn, P. Ranjan, S. Schelter, and Ö. Yılmazeł, Apache mahout: Machine learning on distributed dataflow systems, *J. Mach. Learn. Res.*, vol. 21, no. 127, pp. 1–6, 2020.
[79] B. Quinto, Introduction to spark and spark MLlib, in *Next-Generation Machine Learning with Spark*, B. Quint, ed. New York, NY, USA: Apress, 2020. pp. 29–96.
[80] S. V. Ranawade, S. Navale, A. Dhamal, K. Deshpande, and C. Ghuge, Online analytical processing on hadoop using apache Kylin, *Int. J. Appl. Inform. Syst.*, vol. 12, pp. 1–5, 2017.
[81] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al., Impala: Scalable distributed deep-RL with importance weighted actor-learner architectures, in *Proc. 35th Int. Conf. Machine Learning*, Stockholm, Sweden, 2018, pp. 1407–1416.
[82] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al., Apache hadoop YARN: Yet another resource negotiator, in *Proc. 4th annual Symp. on Cloud Computing*, Santa Clara, CA, USA, 2013, p. 5.
[83] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, Mesos: A platform for Fine-Grained resource sharing in the data center, in *Proc. 8th USENIX Conf. Networked Systems Design and Implementation*, Boston, MA, USA, 2011, pp. 295–308.
[84] S. Wadkar and M. Siddalingaiah, Apache ambari, in *Pro Apache Hadoop*, S. Wadkar and M. Siddalingaiah, eds. Berkeley, CA, USA: Springer, 2014. pp. 399–401.
[85] F. Junqueira and B. Reed, *ZooKeeper: Distributed Process Coordination*. Sebastopol, CA, USA: O'Reilly Media, 2013.
[86] A. Y. Zomaya, *Parallel and Distributed Computing Handbook*. New York, NY, USA: McGraw-Hill Professional, 1995.
[87] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge, UK: Cambridge University Press, 2011.
[88] D. Borthakur, The hadoop distributed file system: Architecture and design, https://www.cs.stolaf.edu/docs/hadoop/hdfs_design.html, 2007.
[89] K. J. Merceedi and N. A. Sabry, A comprehensive survey for Hadoop distributed file system, *Asian J. Res. Comput. Sci.*, vol. 11, no. 2, pp. 46–57, 2021.
[90] J. Dean and S. Ghemawat, MapReduce: A flexible data processing tool, *Commun. ACM*, vol. 53, no. 1, pp. 72–77, 2010.
[91] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, Haloop: Efficient iterative data processing on large clusters, *Proc. VLDB Endow.*, vol. 3, nos. 1&2, pp. 285–296, 2010.
[92] M. Yoon, H. I. Kim, D. H. Choi, H. Jo, and J. W. Chang, Performance analysis of mapReduce-based distributed systems for iterative data processing applications, in *Mobile, Ubiquitous, and Intelligent Computing*, J. J. J.H. Park, H. Adeli, N. Park, and I. Woungang, eds. Berlin, Germany: Springer, 2014. pp. 293–299.
[93] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, The haLoop approach to large-scale iterative data analysis, *VLDB J.*, vol. 21, no. 2, pp. 169–190, 2012.
[94] S. B. Sriramoju, A review on processing big data, *Int. J. Innovat. Res. Comput. Communicat. Eng.*, vol. 2, pp. 2672–2685, 2014.
[95] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in *Proc. 9th USENIX Conf. Networked Systems Design and Implementation*, San Jose, CA, USA, 2012, pp. 15–28.
[96] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, Spark: Cluster computing with working sets, in *Proc. 2nd USENIX Conf. Hot Topics in Cloud Computing*, Boston, MA, USA, 2010, p. 10.
[97] S. Salloum, J. Z. Huang, and Y. He, Random sample partition: A distributed data model for big data analysis, *IEEE Trans. Industr. Inform.*, vol. 15, no. 11, pp. 5846–5854, 2019.
[98] S. Salloum, J. Z. Huang, and Y. He, Empirical analysis of asymptotic ensemble learning for big data, in *Proc. IEEE/ACM 3rd Int. Conf. Big Data Computing Applications and Technologies*, Shanghai, China, 2016, pp. 8–17.
[99] C. Wei, J. Zhang, T. Valiullin, W. Cao, Q. Wang, and H. Long, Distributed and parallel ensemble classification for big data based on kullback-leibler random sample partition, in *Proc. Int. Conf. Algorithms and Architectures for Parallel Processing*, New York, NY, USA, 2020, pp. 448–464.
[100] C. Wei, S. Salloum, T. Z. Emara, X. Zhang, J. Z. Huang, and Y. He, A two-stage data processing algorithm to generate random sample partitions for big data analysis, in *Proc. 11th Int. Conf. Cloud Computing*, Seattle, WA, USA, 2018, pp. 347–364.
[101] T. Z. Emara and J. Z. Huang, A distributed data management system to support large-scale data analysis, *J. Syst. Softw.*, vol. 148, pp. 105–115, 2019.
[102] T. Z. Emara and J. Z. Huang, Distributed data strategies to support large-scale data analysis across geo-distributed data centers, *IEEE Access*, vol. 8, pp. 178526–178538, 2020.
[103] V. Kalavri, V. Brundza, and V. Vlassov, Block sampling: Efficient accurate online aggregation in MapReduce, in *Proc. 2013 IEEE 5th Int. Conf. Cloud Computing Technology and Science*, Bristol, UK, 2013, pp. 250–257.
[104] G. Cormode and N. Duffield, Sampling for big data: A tutorial, in *Proc. 20th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, New York, NY, USA, 2014, p. 1975.
[105] P. Sanders, S. Lamm, L. Hübschle-Schneider, E. Schrade, and C. Dachsbacher, Efficient parallel random sampling-vectorized, cache-efficient, and online, *ACM Trans. Mathemat. Softw.*, vol. 44, no. 3, p. 29, 2018.

[106] E. Gavagsaz, A. Rezaee, and H. H. S. Javadi, Load balancing in reducers for skewed data in MapReduce systems by using scalable simple random sampling, *J. Supercomput.*, vol. 74, no. 7, pp. 3415–3440, 2018.
[107] O. Sagi and L. Rokach, Ensemble learning: A survey, *Wiley Interdiscip Rev Data Min. Knowl Discov*, vol. 8, no. 4, p. e1249, 2018.
[108] J. K. Kim and Z. Wang, Sampling techniques for big data analysis, *Int. Stat. Rev.,* vol. 87, no. S1, pp. S177–S191, 2019.
[109] X. Meng, Scalable simple random sampling and stratified sampling, in *Proc. 30th Int. Conf. Int. Conf. Machine Learning*, Atlanta, GA, USA, 2013, pp. III-531–III-539.
[110] M. S. Mahmud, J. Z. Huang, S. Salloum, T. Z. Emara, and K. Sadatdiynov, A survey of data partitioning and sampling methods to support big data analysis, *Big Data Min. Anal.*, vol. 3, no. 2, pp. 85–101, 2020.
[111] S. Chaudhuri, G. Das, and U. Srivastava, Effective use of block-level sampling in statistics estimation, in *Proc. 2004 ACM SIGMOD Int. Conf. Management of data*, Paris, France, 2004, pp. 287–298.
[112] G. Cormode, M. Garofalakis, P. J. Haas, C. Matthew, Synopses for massive data: Samples, histograms, wavelets, sketches, *Foundat. Trends Databases*, vol. 4, no. 1–3, pp. 1–294, 2012.
[113] S. Salloum, J. Z. Huang, and Y. He, Exploring and cleaning big data with random sample data blocks, *J. Big Data*, vol. 6, no. 1, p. 45, 2019.

Xudong Sun is currently a PhD student at Shenzhen University, China. He received the MS degree in computer science and technology from Shenzhen University, China in 2016 and the BS degree in software engineering from Tianjin Normal University, China in 2011. In 2019, he was awarded the Outstanding Graduate Student of Shenzhen University. His current research interests include big data mining, distributed and parallel computing, and distributed big data approximate computing.

Yulin He received the PhD degree from Hebei University, Baoding, China, in 2014. From 2013 to 2014, he has served as a research assistant with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China. From 2014 to 2017, he worked as a post-doctoral fellow in the College of Computer Science & Software Engineering, Shenzhen University, Shenzhen, China. He is currently a research associate with Big Data Institute of Shenzhen University, Shenzhen, China. His main research interests include big data approximate computing, multi-sample statistic and analysis, and algorithms and application of data mining and machine learning. He has published over 80 research papers in international conferences and journals, including PAKDD, IJCNN, DASFAA, ICPP, CAAI Transactions, ACM Transactions, IEEE Transactions, Elsevier INS, Elsevier PR, Springer APIN, Springer NCAA, etc. He is a CCF number, CIE member, ACM member, IEEE member, and Editorial Review Board members of several international journals.

Dingming Wu is an associate professor with College of Computer Science & Software Engineering at Shenzhen University, China. She received the BS degree in computer science at Huazhong University of Science and Technology, Wuhan, China in 2005, and the MS degree in computer science at Peking University, Beijing, China in 2008. She received the PhD degree in computer science at Aalborg University, Denmark in 2011. Her research concerns data management, query processing, and data mining.

Joshua Zhexue Huang is a distinguished professor of College of Computer Science & Software Engineering at Shenzhen University, China. Also, he is the director of Big Data Institute and the deputy director of the National Engineering Laboratory for Big Data System Computing Technology. He received the PhD degree from Royal Institute of Technology, Sweden in 1993. His main research interests include big data technology and applications. He has published over 200 research papers in conferences and journals. In 2006, he received the first PAKDD most influential paper award. He is known for his contributions to the development of a series of k-means type clustering algorithms in data mining, such as k-modes, fuzzy k-modes, k-prototypes, and w-k-means that are widely cited and used, and some of which have been included in commercial software. He has extensive industry expertise in business intelligence and data mining and has been involved in numerous consulting projects in Australia and China.