BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 01/05 pp85–101
Volume 3, Number 2, June 2020
DOI: 10.26599/BDMA.2019.9020015

A Survey of Data Partitioning and Sampling Methods to Support Big Data Analysis

Mohammad Sultan Mahmud, Joshua Zhexue Huang, Salman Salloum*, Tamer Z. Emara, and Kuanishbay Sadatdiynov

Abstract: Computer clusters with the shared-nothing architecture are the major computing platforms for big data processing and analysis. In cluster computing, data partitioning and sampling are two fundamental strategies to speed up the computation of big data and increase scalability. In this paper, we present a comprehensive survey of the methods and techniques of data partitioning and sampling with respect to big data processing and analysis. We start with an overview of the mainstream big data frameworks on Hadoop clusters. The basic methods of data partitioning are then discussed including three classical horizontal partitioning schemes: range, hash, and random partitioning. Data partitioning on Hadoop clusters is also discussed with a summary of new strategies for big data partitioning, including the new Random Sample Partition (RSP) distributed model. The classical methods of data sampling are then investigated, including simple random sampling, stratified sampling, and reservoir sampling. Two common methods of big data sampling on computing clusters are also discussed: record-level sampling and block-level sampling. Record-level sampling is not as efficient as block-level sampling on big distributed data. On the other hand, block-level sampling on data blocks generated with the classical data partitioning methods does not necessarily produce good representative samples for approximate computing of big data. In this survey, we also summarize the prevailing strategies and related work on sampling-based approximation on Hadoop clusters. We believe that data partitioning and sampling should be considered together to build approximate cluster computing frameworks that are reliable in both the computational and statistical respects.

Key words: big data analysis; data partitioning; data sampling; distributed and parallel computing; approximate computing

1 Introduction

An overwhelming volume of data is now being generated from business transactions, computer simulations, mobile devices, sensors, satellites, social media, and so on. This massive quantity of data can be used to produce high-value information for decision-support, forecasting, business intelligence, research on data-intensive science, and other fields of application. Traditional technologies, such as data warehousing and Structured Query Language (SQL)-based Relational DataBase Management Systems (RDBMSs), have become impractical for handling such a tremendous volume and complexity of big data[1–3]. It is hard or impossible to use a single machine to analyze terabyte-scale datasets, so scalable distributed computing architectures have become a common design choice

• Mohammad Sultan Mahmud, Joshua Zhexue Huang, Salman Salloum, Tamer Z. Emara, and Kuanishbay Sadatdiynov are with National Engineering Laboratory for Big Data System Computing Technology, Shenzhen University, Shenzhen 518060, China, and Big Data Institute, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China. E-mail: {sultan, zx.huang, ssalloum, tamer, kuanishbay}@szu.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2019-05-30; revised: 2019-09-23; accepted: 2019-09-25

for big data analysis frameworks. In these frameworks, data partitioning and sampling are two fundamental strategies for scaling-out and speeding-up big data analysis algorithms. The survey presented in this paper gives a concise summary of the most common methods of partitioning and sampling to support big data analysis on Hadoop clusters.

"Big data" is a label used when the size of the data itself becomes part of the problem. A common strategy for big data analysis on computing clusters is divide-and-conquer[3, 4]. The MapReduce computing model[5] is used to apply this strategy in the mainstream big data analysis frameworks[6–9], such as Apache Hadoop (http://hadoop.apache.org/) and Apache Spark (http://spark.apache.org/). These frameworks implement a shared-nothing architecture (https://www.oreilly.com/learning/processing-data-in-hadoop) where each node is independent in terms of both data and resources. On Hadoop clusters, the Hadoop Distributed File System (HDFS)[10] organizes and replicates a big data file as small distributed data blocks. Studies have shown that when the data size is large enough, parallelization based on distributed data blocks can result in a linear speed-up as computing resources increase in the cluster[11]. In fact, cluster computing frameworks can be scaled easily by adding more machines to the computing cluster. However, the growth rate of data may quickly exceed the available resources. Furthermore, scaling-out a computing cluster requires additional costs and the necessary investment may not be always available in practice[12].

A promising approach to reduce the cost of cluster computing and increase the efficiency of big data analysis is approximate computing[13–17], which uses only a subset of the input data to produce approximate results while achieving low latency and efficient resource utilization[18–20]. Over the past decade, sampling-based approximation techniques have been applied for Approximate Query Processing (AQP) and the statistical analysis of big data on computing clusters[16, 21–23]. In addition, sampling techniques are essential in exploratory data analysis, statistical estimation, and predictive modeling[24, 25]. Nevertheless, sampling from big data is a challenge when considering the block-based organization and the high costs of memory, I/O, and communication on computing clusters with a shared-nothing architecture.

Given the impact of data partitioning and sampling methods on the efficiency and effectiveness of sampling-based approximate big data analysis, we present a concise overview of these methods with respect to big data on Hadoop clusters. On Hadoop clusters, big data is partitioned into small data blocks in HDFS. HDFS blocks are the units of storage, transmission, and processing. Consequently, the partitioning strategy affects the performance of any operation, including sampling. For instance, conducting record-level sampling on HDFS files with many HDFS blocks requires loading the entire data into memory and launching many map tasks (equal to the number of HDFS blocks in the file) to select records from all of the blocks. On the other hand, without considering the statistical properties of the data, the sequential partitioning in HDFS means that there is no guarantee that HDFS blocks are random samples. Consequently, block-level sampling of HDFS files may produce biased results. Thus, the partitioning and sampling strategies should be considered to guarantee the quality of approximate results from HDFS data. In this paper, our objective is to help researchers to get started with data sampling and partitioning to support approximate big data analysis. We consider only structured big data stored in HDFS and focus on the volume dimension of big data.

The remainder of this paper is organized as follows. Section 2 gives an overview of big data analysis frameworks on cluster computing. The most common data partitioning and sampling techniques are given in Sections 3 and 4, respectively. An emerging paradigm for big data analysis on computing clusters is presented in Section 5. The challenges of big data partitioning and sampling for approximate cluster computing are discussed in Section 6. Finally, Section 7 concludes this survey.

2 Big Data Analysis Frameworks

In this section, we start with an overview of cluster computing for big data analysis. We then briefly describe two most common big data analysis frameworks, Apache Hadoop and Apache Spark.

2.1 Overview of cluster computing for big data analysis

To cope with the ever-increasing data volume in a range of different application areas, cluster computing with a shared-nothing architecture has become a common paradigm for building big data analysis frameworks[6, 26, 27]. In a shared-nothing architecture,

each node in the computing cluster is independent in terms of both data and computation. The MapReduce computing model[5,28] is the underlying model in the mainstream big data analysis frameworks[6,7]. A big data file is divided into small non-overlapping data blocks and distributed on the nodes of the computing cluster with HDFS. These blocks are then processed with a parallel, distributed algorithm with two general operations: Map and Reduce. The Map operation processes the distributed data blocks independently and the Reduce operation integrates the Map results to produce the global result for the entire dataset. Figure 1 illustrates the MapReduce model.

Big data technologies include distributed file systems[29,30], distributed computational systems[31], and Massively Parallel Processing (MPP) systems[32,33]. Distributed file systems, such as Google File System (GFS)[34], HDFS[10], and Microsoft Cosmos[35], provide scalable and fault-tolerant storage solutions. Recent advances in these frameworks (e.g., MapReduce[5,28], Hadoop[10], and Cosmos/Dryad[35,36]) have simplified the development of large-scale and the distributed data-intensive applications. Moreover, higher-level programming languages and conceptual data models have been proposed, such as Scope[36], DryadLINQ[37], Pig[38], Dremel[39], Hive[40], Jaql[41], and Tenzing[42].

Hadoop-based computing clusters have become the norm for big data management and analysis in a range of different application areas. Apache Hadoop and Apache Spark are two most widely used big data analysis frameworks in both academia and industry[6–9]. Next, we provide a brief description of these two frameworks before discussing big data partitioning and sampling.

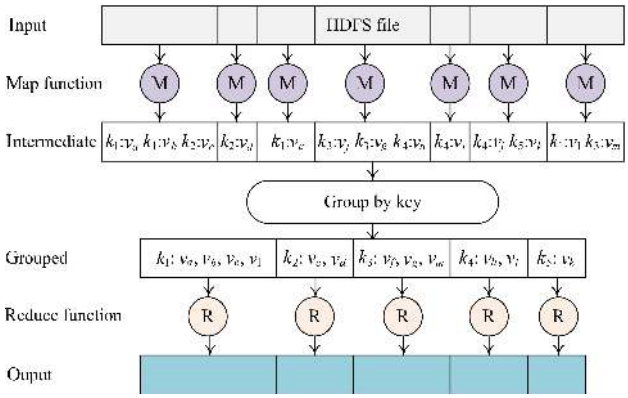

Fig. 1 MapReduce model.

2.2 Apache Hadoop

Apache Hadoop is one of the most well-established platforms supporting the distributed and parallel processing of massive data. It provides a general partitioning mechanism to distribute aggregation workload across different machines using the MapReduce computing model. It is a multi-purpose engine, but not a real-time and high-performance engine because of the high throughput latency in its implementations. The Hadoop platform contains the Hadoop kernel, Hadoop MapReduce, HDFS, the resource manager (YARN), and a number of projects (e.g., Hive and HBase). The Hadoop MapReduce framework[5,28] provides a highly efficient and reliable programming environment for processing large volume distributed datasets.

2.3 Apache Spark

Apache Spark[43] is another open-source and large-scale data processing framework. Spark introduced the core abstraction, Resilient Distributed Dataset (RDD)[44], for distributed in-memory data-parallel computing. RDDs are read-only, immutable, and fault-tolerant collections of elements (objects) distributed or partitioned across a set of nodes in a cluster. RDD supports two types of operations: transformations and actions. Transformations (e.g., map() and filter()) are deterministic but lazy operations that define new RDDs without immediately computing them. Actions (e.g., reduce(), count(), and collect()), on the other hand, launch the computation on RDDs and then return the output to the driver program or store it in a persistent storage system. For more details, see Fig. 2, in which A, B, C, D, E, F, and G are RDDs.

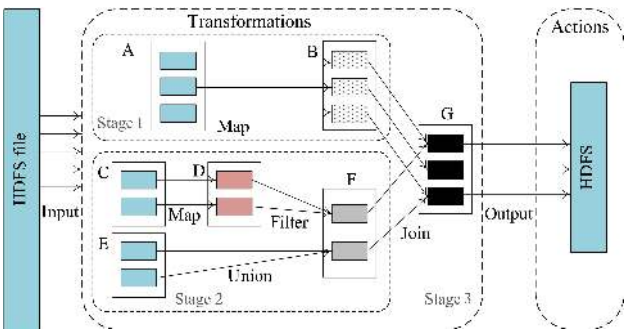

Fig. 2 Data sharing using Spark RDD.

3 Data Partitioning

Data partitioning is a fundamental operation in distributed systems to manage and process big data on computing clusters. In this section, we first briefly review the basic methods of data partitioning. Then, we elaborate on horizontal partitioning and discuss its classical schemes. Finally, we discuss big data partitioning on Hadoop clusters.

3.1 Overview of data partitioning

The purpose of data partitioning is either query processing in databases systems or data-intensive computing in big data analysis frameworks. It was first used in centralized databases[45–49]. Das et al.[50] and Baker et al.[51] investigated solutions for distributed database systems. A workload-aware partition was proposed by Kamal et al.[52] Partitioning solutions for big data applications on NoSQL data-stores were also proposed in Refs. [52, 53]. As Fig. 3 shows, there are three major categories of data partitioning methods: horizontal, vertical, and functional partitioning.

In *horizontal partitioning*, the records of the dataset are divided into disjoint subsets where each subset has the same columns as the entire dataset (see Refs. [48, 50, 54, 55] for more details). This idea started from parallel database systems, and is also known as sharding. Notable implementations of horizontal partitioning are Apache HBase (https://hbase.apache.org/), IBM Informix (https://www.ibm.com/analytics/informix), MongoDB (https://docs.mongodb.com/), MySQL Cluster (https://www.mysql.com/products/cluster/), MySQL (https://www.mysql.com/), Oracle NoSQL database (https://www.oracle.com/technetwork/database/nosqldb/), Spanner (https://cloud.google.com/spanner/), and Teradata (https://www.teradata.com/). There are various schemes for horizontal partitioning, including range, hash, and random schemes. In *vertical partitioning*, the columns of the dataset are divided into subsets that share a key column (see Refs. [45, 46, 53, 56–58] for more details). The columns are divided according to their pattern of use. For example, frequently accessed columns might be placed in one vertical partition and less frequently accessed fields in another partition. Vertical partitioning methods can be classified into two subcategories: optimal solution under restrictive assumptions and heuristic approach. Horizontal and vertical partitioning can also be combined to divide the dataset according to the target application or workload; that is called hybrid partitioning. For example, an e-commerce system might divide the data into two separate subsets: one to store invoice data and the other to store product inventory data. This is sometimes known as *functional partitioning*[59] and is used to improve isolation and data access performance, such as by separating read-write data from read-only data.

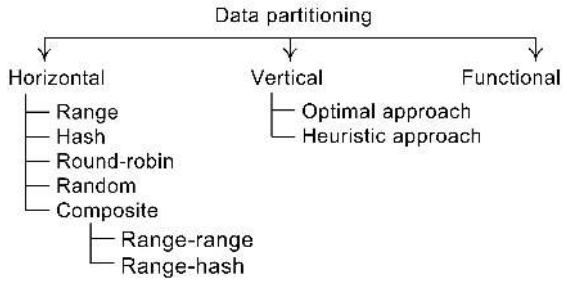

Fig. 3 Data partitioning methods.

3.2 Horizontal data partitioning

In this paper, we focus on horizontal partitioning to distribute datasets with a large number of records. There are three main horizontal partitioning schemes that are commonly used for big data on computing clusters: hash, range, and random. In addition, there is a special kind of hash partition called a round-robin partition. Figure 4 shows how a dataset can be divided with each of these schemes.

- **Hash partitioning**: In a hash partition, records are divided into subsets by hashing the record key and mapping the hash value of the key to a partition. There are multiple methods for this mapping. A common

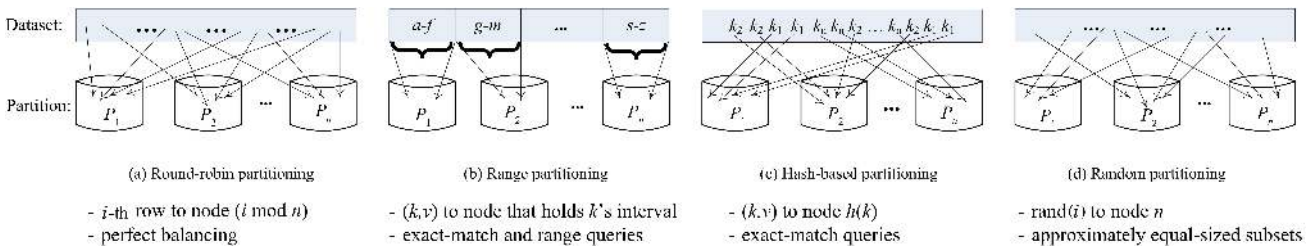

Fig. 4 Horizontal partitioning schemes.

method is a round-robin, which is to mod the hash key with the number of partitions, with the result being the partition ID (with 0 as the first index). It is important to understand that hash-based partition provides a key-wise independent guarantee, because if records have the same key value then they must have the same hash value. Thus, they will be mapped to the same partition. However, a hash-based partition does not guarantee order among partitions. The round-robin method guarantees a balanced partitioning, in which subsets are equivalent in size.

• **Range partitioning**: Range partitioning segments data according to a prescribed range over consecutive ranges of the underlying dataset, and is one of the best partitioning methods to use when global order is required. It provides both a key-wise independent guarantee and partition-wise ordering. Therefore, anything that could be implemented using a hash-based partition can also be implemented using a range partition. Range partitioning requires a set of key-ranges to be predefined. In a distributed environment, how to choose partition boundaries is a challenge. It is especially difficult for massive scale data analysis because typically no statistics about the key distribution over the machines are available at the beginning of the partition. Range partitioning therefore requires a cost-effective and accurate way to determine the partition boundaries and involves a tradeoff of accuracy and cost.

• **Random partitioning**: In a random partition, the records are divided randomly into subsets using a random number generator to determine where to put each record. While random partitioning can produce approximately equal-sized subsets, similar to round-robin, it requires extra processing to calculate a random value for each record.

Table 1 shows a comparison of different horizontal partitioning schemes. Each scheme has its advantages and drawbacks regarding the performance of workload in responsiveness, storage, and processing cost. Also, many works have presented a blended approach to partitioning, for example Ref. [59]. The random and round-robin partitioning methods provide a guarantee of balanced partitions.

3.3 Big data partitioning on Hadoop clusters

Data partitioning is a key issue in big data analysis frameworks. It is used to control the parallelism and achieve scalability to large computing clusters. However, it is a computationally expensive operation when working with big data[60]. In fact, the efficiency and effectiveness of big data queries and analysis algorithms are greatly affected by the data partitioning scheme[60–66]. On Hadoop clusters, data partitioning is basically the responsibility of HDFS[10]. When importing a big data file into HDFS, the file is sequentially divided into small blocks of a fixed storage size determined with byte range. In Apache Spark, the initial step is importing HDFS blocks into the RDD in-memory data structure. An RDD can be partitioned and repartitioned using different methods, such as hash, range, and custom partitioning. Range partitioning is also applied in BerkeleyDB (https://www.oracle.com/database/berkeley-db/), HBase (https://hbase.apache.org/), and MongoDB (https://docs.mongodb.com/), whereas hash-based partitioning is applied in CouchDB (http://couchdb.apache.org/), Clustrix (http://www.clustrix.com), DynamoDB (https://aws.amazon.com/dynamodb/), Riak (http://basho.com/products/riak-kv/), VoltDB (http://voltdb.com/overview), and many other data stores.

Table 1 A comparison of data partitioning schemes.

Scheme	Strength	Limitation
Round-robin	– Sequential scan of the entire dataset – Well-balanced data partition	– Both point and range queries are complicated to process.
Hash	– Sequential scan of the entire dataset – Best suited for point queries based partitioning attributes (only one partition has to be searched)	– Not well-suited for range queries – Also, not well-suited for point queries on non-partitioning attributes
Range	– Sequential scan of the entire dataset – Well-balanced data partition – Well-suited for both point and range queries (only one or few partitions has to be searched)	– Execution skew might occur because of all processes in one or a few partitions.
Random	– Sequential scan of the entire dataset – Approximately balanced data partition	– It requires extra processing to calculate random values. – Records are distributed in random, and no order is followed.

In big data exploration and analysis, we can look at data partitioning as a preprocessing step that prepares the data for subsequent exploration and analysis tasks. For these tasks, the statistical properties of the data should be considered when partitioning big data in order to guarantee the accuracy of the results. A key problem with data partitioning on Hadoop clusters is that HDFS does not consider these statistical properties, such as the probability distribution. For instance, sequentially dividing a big dataset into small data blocks in HDFS does not guarantee that each block is a random sample in the case that the data are not randomly ordered in the original dataset. In such case, using HDFS blocks directly to estimate statistics and build models may lead to statistically incorrect or biased results. Another key issue is data skew, which describes the uneven distribution of the records leading to tasks with different execution times[67–69]. On computing clusters, the performance strongly depends on how evenly data are distributed among the nodes. In fact, this may happen on both the Map side, due to imbalanced input data, and the Reduce side, due to imbalanced intermediate data[70–72]. Sampling has been employed with data partitioning to alleviate the effect of data skew and guarantee load balancing[60, 73], as we discuss in Section 4.3. There is also the problem of imbalanced data, which is a major challenge to machine learning algorithms[74]. Classical data partitioning methods do not consider the class or key distribution.

While range and hash partitioning are the most common methods, random partitioning is necessary to guarantee that the data is uniformly distributed across the nodes. One work on distributed data randomization on Hadoop clusters is Cloud OnLine Aggregation (COLA)[75]. It introduces a preprocessing stage to randomize data in HDFS using a MapReduce job. In the Map operation, a random number between 1 and P (the number of HDFS blocks) is assigned to each record in the data. Then, each record is written to the assigned block in the Reduce operation. After that, block-level sampling can be used for online aggregation by sequentially reading the randomized blocks from HDFS. In addition to the general random partitioning scheme, sampling-based data partitioning is required to make the distributed data blocks reflect the statistical properties of the entire dataset. This is done, for instance, by making each HDFS block a simple random sample or stratified random sample of the entire data. A promising work in this direction is the Random Sample Partition (RSP)[76], which is a distributed data model to represent a big dataset as a set of non-overlapping data blocks, called RSP blocks. Each RSP block is a random sample of the entire dataset. An RSP can be generated from an HDFS file using a two-stage data partitioning method[77, 78]. Each RSP block is created by combining approximately equal random slices from all of the original HDFS blocks.

As Fig. 5 shows, RSP blocks preserve the probability distribution of the entire dataset. This partitioning operation can be scheduled to run offline on the computing cluster. An RSP is saved as an HDFS-RSP file with metadata storing the RSP block information, including the size and location. An RSP-based Big Data Management System (BDMS) has been formulated by Emara and Huang[78]. Moreover, an open source of a Spark library to represent HDFS blocks as a set of RSP blocks has been developed in Ref. [79]. Selecting an RSP block from an HDFS-RSP file is equivalent to drawing a random sample directly from the original HDFS. The RSP model reduces the sampling time from hours to seconds on small computing clusters. Consequently, data scientists can use RSP blocks directly in sampling-based approximate big data analysis. This solves a key issue on Hadoop clusters because completely random disk access can be five orders of magnitude slower than sequential access[80].

4 Data Sampling

Sampling is an essential strategy to reduce the burden of big data volume. In this section, we start with an overview of the random sampling strategy commonly used in data science. Then, we briefly review four key sampling schemes. After that, we discuss big data sampling on Hadoop clusters.

4.1 Overview of data sampling

The goal of random sampling is to obtain representative small subsets that can be processed efficiently to explore and analyze the data[81, 82]. Data scientists often use small random samples to obtain sample statistics,

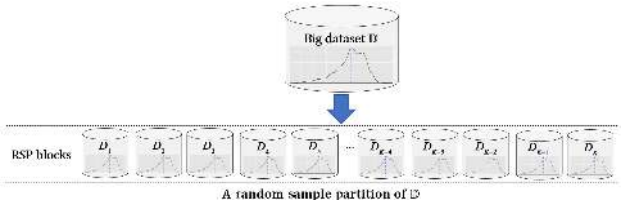

Fig. 5 Random sample partition distributed data model.

assess the quality of estimators, and test statistical models with different algorithms. Random sampling is also fundamental in statistical estimation procedures, such as the bootstrap[83]. In fact, random sampling has a long history of use in databases[84, 85], but it is becoming more important in the big data era, when handling an entire dataset all at once may not be possible considering the ever-increasing volume of data. Data quality is often more important than data quantity when using a sample to make an estimate or build a model[86]. In addition, using small random samples allows for greater attention to data quality and enables deeper data exploration[87]. Furthermore, random sampling is essential in approximate and incremental computing[81, 88], as we discuss in Section 5.

Many researchers have considered sampling-based approaches to estimate various statistics in the context of database systems and data streams[89–94]. Rojas et al.[95] suggested data exploration on smaller but better-selected samples generated from sampling techniques other than random sampling. Kandel et al.[96] pointed out that the data scientists they interviewed were concerned about using data sampling in big data, because of the bias it could introduce into their analysis. In the same vein, Lin and Ryaboy[97], who noted that it is easy to make errors when sampling from a big dataset and that it runs contrary to the objective of big data analysis, also suggested using as much data as possible and running experiments at scale.

4.2 Data sampling methods

There are a range of different methods to draw a sample from a dataset. Choosing between these methods, which are known as sampling schemes, depends mainly on the target application or workload. In the following, we introduce key sampling schemes and methods commonly used in data science. Table 2 summarizes the sampling methods.

• **Bernoulli sampling:** In this scheme, each item in the dataset has an equal probability to be included in the sample[98]. This sampling scheme operates without replacement, where each data item is selected independently for the sample[99]. Consequently, the sample size is random and not fixed. Thus, it is difficult to estimate the processing latency over Bernoulli samples. To overcome this limitation, simple random sampling scheme can be used.

• **Simple random sampling:** This is a flexible and general method for constructing a synopsis of data items, and is one of the most common sampling techniques[100–102]. In this scheme, each data item has an equal chance to be included in the selected sample. Simple random sampling performed with replacement allows each data item to appear multiple times in the sample; sampling without replacement allows each data item to appear at most one time. However, simple random sampling does not ensure that each group in the original data is considered fairly in the sample. Stratified sampling can be used to overcome this limitation.

• **Stratified sampling:** This is a sampling scheme in which the original data is divided into a homogeneous disjoint set of groups (strata); from each group (stratum) a random sample is drawn and these are combined to build the sample of the original data[103]. Stratified sampling ensures that data items from each group are considered fairly in the sample and no group will be overlooked. Compared to simple random sampling, stratified sampling provides higher statistical precision and reduces the sampling error. It also requires a smaller sample size to achieve the same accuracy as simple random sampling, thus further improving performance and utilizing less computing resources.

• **Reservoir sampling:** This is done without replacement from a big array (list) in a single pass, where the length of the array is indeterminate or unbounded[104]. Reservoir sampling receives data items from an array and maintains a sample in a buffer called a *reservoir*. If the dataset consists of an unknown number of items, or too many to fit into storage, then simple random sampling does not work, and reservoir sampling can be used. The reservoir sampling technique has been used extensively in large-scale data mining applications (see Refs. [105–107] for more details).

Table 2 A summary of common sampling methods.

Method	Description
Simple random sampling[100–102]	Data items are selected with equal probability and the sample size is fixed.
Bernoulli sampling[98, 99]	Data items are selected with equal probability but the sample size is random.
Stratified sampling[103]	Data items are divided into strata and a sample is drawn from each stratum.
Reservoir sampling[104]	Data items are added to a reservoir of a fixed size.
Bootstrapping[83, 108]	Multiple samples are drawn with replacement and used for statistical estimation and diagnosis.

• **Bootstrap method:** This is a classical method to assess the variability of a sample statistic. It uses multiple samples with replacement from the observed dataset[83, 109]. However, bootstrapping on big datasets requires high computational and storage costs as it depends on repeatedly drawing samples of sizes comparable to the original dataset and computing estimates from all these samples.

4.3 Big data sampling on Hadoop clusters

To employ the previous sampling methods on Hadoop clusters, we need to consider the special block-based architecture of cluster computing frameworks. Thus, we can discuss two common ways to get random samples from big datasets stored in HDFS depending on whether the sampling units are individual records or blocks of records.

• **Record-Level Sampling (RLS)**: This depends on the random selection of individual records from the dataset, and is an expensive operation in cluster computing frameworks that implement a shared-nothing architecture[60]. Record-level sampling goes through all of the records sequentially, and is thereby highly time-consuming. It requires a complete pass over the entire distributed big dataset and results in communication and I/O costs. For a large HDFS file, record-level sampling is not efficient because it reads the data record-by-record. This operation becomes more challenging when many disjoint random samples are required, as in ensemble methods[24, 110] and statistical estimation methods like the bootstrap. The Bag of Little Bootstraps (BLB) is one approach to scale the classical bootstrap method to big data by drawing samples of small sizes[108, 111]. However, obtaining small disjoint random samples is still a challenge in cluster computing frameworks[112]. Spark supports sampling on RDDs. In particular, Spark's sampling functions can be classified into two categories: simple random sampling using the sample() function and stratified sampling using sampleByKey() and sampleByKeyExact(). Spark implements these sampling functions in a batch fashion, where all data items are first accumulated in a batch, and then the actual sampling is carried out. In addition, the divide-and-conquer strategy is used to scale sampling algorithms to big data on computing clusters, as in Refs. [113, 114]. Sampling is also used as fundamental strategy to solve the data skew problem on Hadoop clusters, as in Ref. [115].

• **Block-Level Sampling (BLS)**: This method considers that a big dataset is stored as a set of disjoint data blocks[116], each containing a small subset of records. In this case, a block instead of a record is randomly selected during the sampling process. Block-level sampling is appealing in cluster computing frameworks, since HDFS data blocks are the units of both storage and processing in these frameworks. In contrast to record-level sampling, block-level sampling requires significantly fewer block accesses for the same sample size. However, the results obtained from block-level samples may be biased or incorrect because the data in HDFS blocks may be correlated. The same problem arises with RDDs in Apache Spark using partition-level sampling. Furthermore, the entire RDD should be read into memory in order to obtain a block-level sample. The RSP model[76] solves these problems by making HDFS blocks into ready-to-use random sample data blocks. RSP blocks have unbiased and consistent estimators. Hence, RSP blocks can be used directly in statistical estimation and predictive modeling, especially when analyzing big data requires more than the available resources to meet specific application requirements.

In the absence of a statistical summary, drawing a random sample from a distributed dataset is a nontrivial task, because we cannot perform sampling arbitrarily. Experts have also suggested that using multiple sampling strategies on the same dataset would enable a more effective evaluation of a dataset. Therefore, this domain is in need of more research to find better solutions.

5 Approximate Cluster Computing for Big Data Analysis

Approximate computing has become a common and necessary paradigm to cope with the ever-increasing data volume on computing clusters. In this section, we first present an overview of approximate computing. Then, we elaborate on the sampling-based approximation approach for big data analysis on Hadoop clusters.

5.1 Overview of approximate computing

Data is growing exponentially, and even faster than Moore's law predicts of computational power. Nowadays, modern services use big data analysis systems to mine and extract valuable patterns and trends from data. Handling these data is quite expensive.

Recently, approximate computing has emerged as a promising solution to reduce the computing resources usage, processing time, and even energy consumption of big data analysis frameworks[13, 14, 19, 20, 22, 117, 118]. Unlike traditional computing, approximate computing is done over a small synopsis of the data instead of the entire dataset. Many data algorithms are amenable to an approximate result rather than an exact one[119].

In fact, it is sometimes impossible to obtain exact results, either due to the underlying algorithm, e.g., machine learning algorithms, or due to the data generation process, given that real-world datasets often have noise that affects the results. Approximate computing makes a trade-off between accuracy and efficiency. This trade-off can be depicted as a "runtime–resources usage–accuracy" triangle as shown in Fig. 6.

Due to the growth of digital data being faster than the growth of computational power, approximate computing is emerging as an essential technique for big data analytics with interactive response times. Approximate computing is sometimes combined with incremental computing where the data is processed incrementally and the results are updated accordingly. This technique is also known as incremental approximate computing, see Fig. 7 for more details.

There are various approximation techniques that have been proposed in databases for approximate query processing, including sampling, sketching, histograms, and online aggregation[21, 116]. These techniques have been recently extended to big data on computing clusters. Sampling, in particular, has been adopted in cluster computing frameworks, as discussed in Section 5.2.

5.2 Sampling-based approximation on Hadoop clusters

Sampling is one of the most commonly used techniques to enable approximation on Hadoop clusters[103, 120]. We introduce the current big data frameworks for sampling-

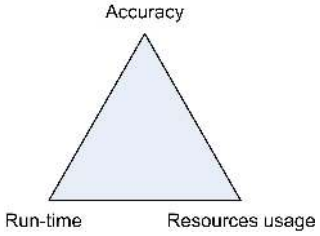

Fig. 6 Approximate computing trades-off accuracy with run-time and resources.

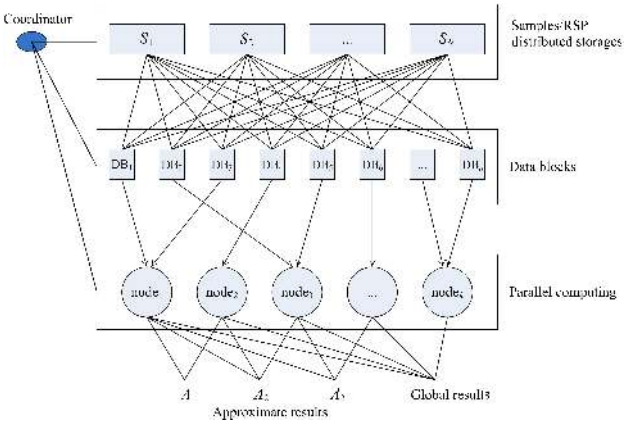

Fig. 7 Distributed approximate computing.

based approximate computing on Hadoop clusters as follows.

- **Early Accurate Result Library (EARL)**[121] is an extension of Hadoop that provides early estimation results. It uses online uniform sampling from HDFS files with the bootstrap method to incrementally evaluate the accuracy.
- **ApproxHadoop**[20] uses multi-stage sampling to enable approximation in Hadoop MapReduce.
- **ApproxSpark**[122] uses multi-stage sampling or adaptive stratified reservoir sampling to enable approximation in Apache Spark. It supports both record-level and block-level sampling (called data item-level and partition-level, respectively in Ref. [122]). However, block-level sampling leads to larger error bounds since the data in the RDD is not necessarily randomized.
- **BlinkDB**[19] is a distributed sampling-based approximate query engine that supports SQL-based aggregation queries with error and time bounds.
- **BlinkML**[123] enables approximate machine learning by training a model on a small sample instead of the entire data and providing error bounds on the accuracy of the approximate model. It supports models that rely on maximum likelihood estimation, such as linear regression, logistic regression, max entropy classifier, and Probabilistic Principal Component Analysis (PPCA). To obtain a sample, BlinkML uses online uniform sampling without replacement. If the dataset cannot fit into memory, BlinkML either uses Bernolli sampling or offline samples from a pre-shuffled dataset.
- **IncApprox**[119] is a stream data analytics system which depends on both approximate and incremental computing to incrementally update an approximate output for data analysis tasks.

• **Sapprox**[124] depends on the distribution of the dataset in the file system for approximation. It collects the occurrences of subsets in an offline preprocessing stage and uses these to facilitate online sampling. It uses cluster sampling with unequal probability to address the data skew problem.

• **RSP approach**[76] is a new approach for approximate big data analysis using the RSP distributed data model. It depends on a step-wise process to analyze data in batches of RSP blocks. Each batch is a block-level sample of RSP blocks that are processed using sequential algorithms in a data-parallel fashion. With this approach, a few RSP blocks are enough to obtain approximate results that are equivalent to those built from the entire dataset. The RSP approach has been applied to different tasks in predictive modeling and exploratory data analysis (see Refs. [125, 126] for more details).

• **ApproxIoT**[127] depends on edge computing resources to enable sampling-based approximation in IoT with an online hierarchical stratified reservoir sampling algorithm.

Table 3 presents a summary of key big data frameworks for sampling based approximate computing.

6 Discussion

Since cluster and approximate computing are two common and necessary paradigms for big data analysis frameworks, efficient and effective data partitioning and sampling techniques are fundamental for big data analysis. In this section, we further discuss the current challenges of enabling approximate cluster computing and scaling algorithms for big data.

• **Bottlenecks of cluster computing in big data analysis**: Although the mainstream big data analysis frameworks employ the data-parallel model to run scalable algorithms on computing clusters, analyzing an entire dataset may exceed the available resources in a computing cluster. Since the rate of data production is outracing technology scaling, the ever-increasing data volume can quickly exceed the memory of a computing cluster. Hadoop MapReduce is efficient for algorithms that scan the entire big dataset once. However, for iterative data analysis algorithms, it becomes inefficient because of heavy I/O and communication costs[128]. Apache Spark with its in-memory computing model is much faster than the disk-based cluster computing in Hadoop MapReduce[43]. Therefore, it is very efficient for iterative algorithms. Nonetheless, if the memory is not large enough to hold all of the data blocks of a big dataset, the computation will dramatically slow down.

• **Sampling-based big data partitioning**: Conventional data partitioning methods (e.g., range and hash) and data partitioning techniques in distributed file systems (e.g., sequential partitioning in HDFS) do not necessarily satisfy the requirements of data analysis tasks. These methods do not consider the statistical properties of the data which may lead to very poor results. Therefore, it is essential to develop statistically-aware data partitioning methods in big data analysis frameworks. This enables the effective and efficient use of HDFS data blocks to obtain approximate results using sequential algorithms. However, high-quality data partitioning is one of the most expensive operations for distributed computing because typically no statistics are available about the data distribution. Big data analysis requires a not only computationally efficient but also statistically effective approach at both the data management and analysis level.

• **Online big data sampling**: It is straightforward to obtain a random sample when data are centralized

Table 3 A summary of frameworks for sampling-based approximation on Hadoop clusters.

Framework	Description
BlinkDB[19]	Approximate distributed query processing engine that uses stratified sampling
ApproxHadoop[119]	Uses multi-stage sampling for approximate MapReduce job execution
EARL[121]	Uses online uniform sampling from HDFS files with the bootstrap method
ApproxSpark[122]	Uses multi-stage sampling or adaptive stratified reservoir sampling
	Supports both record-level and block-level sampling
BlinkML[123]	Uses online uniform sampling without replacement
	If the dataset cannot fit into the memory, either uses Bernoulli sampling or offline samples
IncApprox[119]	A stream data analytics system which depends on both approximate and incremental computing
Sapprox[124]	Uses it to facilitate online sampling
RSP approach[76]	Approach for approximate big data analysis
	Depends on a step-wise process to analyze data in batches of RSP blocks
ApproxIoT[127]	Sampling-based approximation in IoT with an online hierarchical stratified reservoir sampling algorithm

and the size is known. However, in the big data era, many applications deal with data that are distributed and unbounded. Drawing a random sample from distributed data becomes difficult for two main reasons. First, when the size of data is unknown, it is not possible to predetermine sampling probability. Second, data are distributed on different machines and it is not feasible to collect it to a central machine for sampling. Combined these challenges give rise to the question of how to obtain a random sample from distributed data efficiently with a guarantee of sample uniformity. Classical sampling techniques (e.g., random) require a full scan of the dataset each time to generate a random sample, and are therefore ineffective and cumbersome given the increasing volume of the data stored in a distributed system. Current big data systems are mainly targeted toward batch processing (data-in-rest). In contrast to classical offline sampling (batches), online sampling from a massively distributed dataset is difficult. In this regard, partitioning a big dataset into small subsets (i.e., data blocks), each being a random sample of the entire dataset, is a fundamental operation for big data analysis.

- **Ensemble methods for big data analysis**: As mentioned above, divide and conquer is a common strategy in current big data analysis frameworks. The big data ensemble model is different from the classical ensemble model. In big data analysis, an ensemble model integrates results of different subsets or samples of data, whereas a classical ensemble combines the results of different models or algorithms on the same dataset to produce a robust result. Therefore, the key question for data partitioning is how to aggregate the results from these subsets. It is theoretically and practically necessary to find appropriate ensemble functions (consensus function) for distributed datasets.
- **Approximate big data analysis**: It is difficult or impractical to process an entire big dataset, especially on small computing clusters. In extreme cases, it is not even possible to store the entire input dataset. Thus, a key research question is whether the entire data needs to be used to find properties and reveal insights, or if a subset is sufficient. To meet the challenge, we may apply incremental approximate computing in the distributed computing cluster to achieve efficiency. Understanding theoretical trade-offs between accuracy and sample size is another important open research issue. With incremental approximate computation, big data can be analyzed incrementally to obtain approximate results that are asymptotically equivalent to those computed using entire dataset. In this way, computational resources and time can be decreased significantly. A reliable combination between cluster computing and approximate computing requires addressing key issues, such as sample selection, sample size, accuracy measures, and aggregation functions.

Considering the impact of sampling and partitioning on the performance and accuracy of sampling-based approximation on Hadoop clusters, we end this section with the following Research Questions (RQs):

RQ1: How can we draw a random sample from a massively distributed dataset on Hadoop clusters, considering that memory may never be sufficient to hold an entire big dataset, since data growth rate is faster than technology scaling?

RQ2: How can we quickly obtain a random sample partition from a big dataset so that data scientists can directly use random sample data blocks to explore and analyze big data using their preferred techniques and libraries?

RQ3: How can we aggregate the local results from random sample data blocks of a big dataset for different data analysis and mining algorithms?

RQ4: How much data (portion size) is sufficient from an entire big dataset to approximate a result that is equivalent to the result from the entire dataset?

7 Conclusion

Data partitioning and sampling can provide tremendous benefits by improving the scalability, manageability, and performance of big data analysis algorithms on computing clusters. In this paper, the partitioning and sampling techniques for big data analysis were reviewed. While the key classical partitioning schemes are employed on computing clusters, new sampling-based partition models have become fundamental to increase scalability. Furthermore, this is essential to guarantee the quality of the selected samples and this to yield more accurate approximate results. In addition to data partitioning and sampling, key projects in sampling-based approximation for big data analysis were briefly reviewed. Also, we highlighted the critical technical challenges of partitioning and sampling to support approximate big data analysis on computing clusters.

Acknowledgment

This research was Supported in part by the National Natural Science Foundation of China (No. 61972261) and the National Key R&D Program of China (No. 2017YFC0822604-2).

References

[1] R. Cattell, Scalable SQL and NoSQL data stores, *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[2] K. Bakshi, Considerations for big data: Architecture and approach, in *Proc. of 2012 IEEE Aerospace Conference*, Big Sky, MT, USA, 2012, pp. 1–7.

[3] X. Chen and M. Xie, A split-and-conquer approach for analysis of extraordinarily large data, *Statistica Sinica*, vol. 24, no. 4, pp. 1655–1684, 2014.

[4] N. Lazar, The big picture: Divide and combine to conquer big data, *Chance*, vol. 31, no. 1, pp. 57–59, 2018.

[5] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, in *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI'04)*, San Francisco, CA, USA, 2004, pp. 137–150.

[6] D. Singh and C. K. Reddy, A survey on platforms for big data analytics, *Journal of Big Data*, vol. 2, no. 1, p. 8, 2014.

[7] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, Big data and its technical challenges, *Communications of the ACM*, vol. 57, no. 7, pp. 86–94, 2014.

[8] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al., Apache spark: A unified engine for big data processing, *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[9] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, Big data analytics on apache spark, *International Journal of Data Science and Analytics*, vol. 1, no. 3, pp. 145–164, 2016.

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, The Hadoop distributed file system, in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, NV, USA, 2010, pp. 1–10.

[11] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, Map-reduce for machine learning on multicore, in *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'06)*, Cambridge, MA, USA, 2006, pp. 281–288.

[12] D. Quoc, Approximate data analytics systems, PhD dissertation, Technische Universität Dresden, Dresden, Germany, 2017.

[13] R. Nair, Big data needs approximate computing: Technical perspective, *Communications of the ACM*, vol. 58, no. 1, pp. 104–104, 2015.

[14] S. Mittal, A survey of techniques for approximate computing, *ACM Computing Surveys*, vol. 48, no. 4, pp. 1–33, 2016.

[15] D. A. Reed and J. Dongarra, Exascale computing and big data, *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[16] C. E. Otero and A. Peter, Research directions for engineering big data analytics software, *IEEE Intelligent Systems*, vol. 30, no. 1, pp. 13–19, 2015.

[17] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura, Approximate computing: Challenges and opportunities, in *Proc. of 2016 IEEE International Conference on Rebooting Computing (ICRC)*, San Diego, CA, USA, 2016, pp. 1–8.

[18] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, Managing performance vs. accuracy trade-offs with loop perforation, in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE'11)*, Szeged, Hungary, 2011, pp. 124–134.

[19] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, BlinkDB: Queries with bounded errors and bounded response times on very large data, in *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys'13)*, Prague, Czech Republic, 2013, pp. 29–42.

[20] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, ApproxHadoop: Bringing approximations to MapReduce frameworks, in *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'15)*, Istanbul, Turkey, 2015, pp. 383–397.

[21] K. Li and G. Li, Approximate query processing: What is new and where to go?, *Data Science and Engineering*, vol. 3, no. 4, pp. 379–397, 2018.

[22] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica, Knowing when you're wrong: Building fast and reliable approximate query processing systems, in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*, Snowbird, UT, USA, 2014, pp. 481–492.

[23] D. L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe, Streamapprox: Approximate computing for stream analytics, in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference (Middleware'17)*, Las Vegas, NV, USA, 2017, pp. 185–197.

[24] O. Sagi and L. Rokach, Ensemble learning: A survey, *Data Mining and Knowledge Discovery*, vol. 8, no. 4, pp. 1–18, 2018.

[25] S. Basiri, E. Ollila, and V. Koivunen, Robust, scalable, and fast bootstrap method for analyzing large scale data, *IEEE Transactions on Signal Processing*, vol. 64, no. 4, pp. 1007–1017, 2016.

[26] 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 *Proceedings of International Conference on Data and Information Systems (ICDIS'17)*, Amarkantak, India, 2017, pp. 231–240.

[27] I. Polato, R. Ré, A. Goldman, and F. Kon, A comprehensive view of Hadoop research–systematic literature review, *Journal of Network and Computer Applications*, vol. 46, pp. 1–25, 2014.

[28] J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[29] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, A survey of peer-to-peer storage techniques for distributed file systems, in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) – Volume II*, Las Vegas, NV, USA, 2005, pp. 205–213.

[30] P. Bzoch and J. Safarik, State of the art in distributed file systems: Increasing performance, in *Proc. of 2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems*, Bratislava, Slovakia, 2011, pp. 153–154.

[31] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, 3rd ed. New York, NY, USA: Springer-Verlag, 2011.

[32] D. Taniar, High performance database processing, in *Proc. of 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, Fukuoka, Japan, 2012, pp. 5–6.

[33] S. Vijayakumar, A. Bhargavi, U. Praseeda, and S. A. Ahamed, Optimizing sequence alignment in cloud using Hadoop and Mpp database, in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing (CLOUD'12)*, Honolulu, HI, USA, 2012, pp. 819–827.

[34] S. Ghemawat, H. Gobioff, and S.-T. Leung, The google file system, *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[35] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, Dryad: Distributed data-parallel programs from sequential building blocks, in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys'07)*, Lisbon, Portugal, 2007, pp. 59–72.

[36] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, Scope: Easy and efficient parallel processing of massive data sets, *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1265–1276, 2008.

[37] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, and J. Currey, Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language, in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI'08)*, San Diego, CA, USA, 2008, pp. 1–14.

[38] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, Pig latin: A not-so-foreign language for data processing, in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, Vancouver, Canada, 2008, pp. 1099–1110.

[39] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, Dremel: Interactive analysis of web-scale datasets, *Communications of the ACM*, vol. 54, no. 6, pp. 114–123, 2011.

[40] A. Thusoo, J. 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 *Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE'10)*, Long Beach, CA, USA, 2010, pp. 996–1005.

[41] K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Eltabakh, C.-C. Kanne, F. Ozcan, and E. J. Shekita, Jaql: A scripting language for large scale semi structured data analysis, *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1272–1283, 2011.

[42] B. Chattopadhyay, L. Lin, W. Liu, S. Mittal, P. Aragonda, V. Lychagina, Y. Kwon, and M. Wong, Tenzing: A SQL implementation on the MapReduce framework, *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1318–1327, 2011.

[43] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analytics*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2015.

[44] 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 *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*, San Jose, CA, USA, 2012, p. 2.

[45] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, Vertical partitioning algorithms for database design, *ACM Transactions on Database Systems*, vol. 9, no. 4, pp. 680–710, 1984.

[46] D. W. Cornell and P. S. Yu, An effective approach to vertical partitioning for physical design of relational databases, *IEEE Transactions on Software Engineering*, vol. 16, no. 2, pp. 248–258, 1990.

[47] W. W. Chu and I. T. Ieong, A transaction-based approach to vertical partitioning for relational database systems, *IEEE Transactions on Software Engineering*, vol. 19, no. 8, pp. 804–812, 1993.

[48] C. Curino, E. Jones, Y. Zhang, and S. Madden, Schism: A workload-driven approach to database replication and partitioning, *Proceedings of the VLDB Endowment*, vol. 3, no. 1, pp. 48–57, 2010.

[49] C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, Relational cloud: A database-as-a-service for the cloud, in *Proc. of 5th Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, 2011, pp. 235–240.

[50] S. Das, D. Agrawal, and A. El Abbadi, Elastras: An elastic transactional data store in the cloud, in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing (HotCloud'09)*, San Diego, CA, USA, 2009, pp. 1–5.

[51] J. Baker, C. Bond, J. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh, Megastore: Providing scalable, highly available storage for interactive services, in *Proc. of Conference on Innovative Database Research (CIDR)*, Asilomar, CA, USA, 2011, pp. 223–234.

[52] J. Kamal, M. Murshed, and R. Buyya, Workload-aware incremental repartitioning of shared-nothing distributed databases for scalable OLTP applications, *Future Generation Computer Systems*, vol. 56, pp. 421–435, 2016.

[53] S. P. Phansalkar and A. R. Dani, Transaction aware vertical partitioning of database (TAVDP) for responsive OLTP applications in cloud data stores, *Journal of Theoretical and Applied Information Technology*, vol. 59, no. 1, pp. 73–81, 2014.

[54] P. A. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. B. Lomet, R. Manne, L. Novik, and T. Talius, Adapting microsoft SQL server for cloud computing, in *Proc. of 2011 IEEE 27th International Conference on Data Engineering*, Hannover, Germany, 2011, pp. 1255–1263.

[55] S. Ahirrao and R. Ingle, Scalable transactions in cloud data stores, in *Proc. of 2013 3rd IEEE International Advance Computing Conference (IACC)*, Ghaziabad, India, 2013, pp. 116–119.

[56] S. B. Navathe and M. Ra, Vertical partitioning for database design: A graphical algorithm, *ACM SIGMOD Record*, vol. 18, no. 2, pp. 440–450, 1989.

[57] J. H. Son and M. H. Kim, An adaptable vertical partitioning method in distributed systems, *Journal of Systems and Software*, vol. 73, no. 3, pp. 551–561, 2004.

[58] W. Zhao, Y. Cheng, and F. Rusu, Workload-driven vertical partitioning for effective query processing over raw data, arXiv preprint arXiv: 1503.08946, 2015.

[59] Y.-F. Huang and C.-J. Lai, Integrating frequent pattern clustering and branch-and-bound approaches for data partitioning, *Information Sciences*, vol. 328, pp. 288–301, 2016.

[60] M. Vojnovic, F. Xu, and J. Zhou, Sampling-based range partition methods for big data analytics, Technical Report MSR-TR-2012-18, Microsoft Research, Redmond, WA, USA, 2012.

[61] A. Chakrabarti, S. Parthasarathy, and C. Stewart, Green- and heterogeneity-aware partitioning for data analytics, in *Proc. of IEEE Conference on Computer Communications Workshops*, San Francisco, CA, USA, 2016, pp. 366–371.

[62] S. Phansalkar and S. Ahirrao, Survey of data partitioning algorithms for big data stores, in *Proc. of 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, Waknaghat, India, 2016, pp. 163–168.

[63] A. Shanbhag, A. Jindal, S. Madden, J. Quiane, and A. J. Elmore, A robust partitioning scheme for ad-hoc query workloads, in *Proceedings of the 2017 Symposium on Cloud Computing (SoCC'17)*, Santa Clara, CA, USA, 2017, pp. 229–241.

[64] J. Wang, Q. Xiao, and J. Yin, DRAW: A new Data-gRouping-AWare data placement scheme for data intensive applications with interest locality, *IEEE Transactions on Magnetics*, vol. 49, no. 6, pp. 2514–2520, 2013.

[65] K. H. K. Reddy and D. S. Roy, DPPACS: A novel data partitioning and placement aware computation scheduling scheme for data-intensive cloud applications, *The Computer Journal*, vol. 59, no. 1, pp. 64–82, 2016.

[66] D. Yuan, Y. Yang, X. Liu, and J. Chen, A data placement strategy in scientific cloud workflows, *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010.

[67] K. Slagter, C.-H. Hsu, Y.-C. Chung, and D. Zhang, An improved partitioning mechanism for optimizing massive data analysis using MapReduce, *The Journal of Supercomputing*, vol. 66, no. 1, pp. 539–555, 2013.

[68] Q. Chen, J. Yao, and Z. Xiao, Libra: Lightweight data skew mitigation in MapReduce, *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 9, pp. 2520–2533, 2015.

[69] M. He, G. Li, C. Huang, Y. Ye, and W. Tian, A comparative study of data skew in Hadoop, in *Proceedings of 26th International Conference on Network, Communication and Computing (ICNCC'17)*, Kunming, China, 2017, pp. 1–6.

[70] Z. Tang, W. Lv, K. Li, and K. Li, An intermediate data partition algorithm for skew mitigation in spark computing environment, *IEEE Transactions on Cloud Computing*, doi: 10.1109/TCC.2018.2878838.

[71] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, A study of skew in MapReduce applications, in *The 5th Open Cirrus Summit*, Moskow, Russia, 2011, pp. 1–5.

[72] Z. Tang, X. Zhang, K. Li, and K. Li, An intermediate data placement algorithm for load balancing in spark computing environment, *Future Generation Computer Systems*, vol. 78, no. 1, pp. 287–301, 2018.

[73] Y. Xu, P. Zou, W. Qu, Z. Li, K. Li, and X. Cui, Sampling-based partitioning in MapReduce for skewed data, in *Proc. of 2012 Seventh ChinaGrid Annual Conference (CHINAGRID'12)*, Beijing, China, 2012, pp. 1–8.

[74] S. del Ro, V. Lpez, J. M. Bentez, and F. Herrera, On the use of MapReduce for imbalanced big data using random forest, *Information Sciences*, vol. 285, pp. 112–137, 2014.

[75] X. Ci and X. Meng, An efficient block sampling strategy for online aggregation in the cloud, in *Proc. of International Conference on Web-Age Information Management (WAIM 2015)*, Qingdao, China, 2015, pp. 362–373.

[76] S. Salloum, J. Z. Huang, Y. He, X. Zhang, T. Z. Emara, C. Wei, and H. He, A random sample partition data model for big data analysis, arXiv preprint arXiv: 1712.04146, 2017.

[77] 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. of International Conference on Cloud Computing (CLOUD 2018)*, Seattle, WA, USA, 2018, pp. 347–364.

[78] T. Z. Emara and J. Z. Huang, A distributed data management system to support large-scale data analysis, *The Journal of Systems and Software*, vol. 148, pp. 105–115, 2019.

[79] T. Z. Emara and J. Z. Huang, RRPlib: A Spark library for representing HDFS blocks as a set of random sample data blocks, *Science of Computer Programming*, vol. 184, pp. 1–7, 2019.

[80] A. Jacobs, The pathologies of big data, *Communications of the ACM*, vol. 52, no. 8, pp. 36–44, 2009.

[81] G. Cormode and N. Duffield, Sampling for big data: A tutorial, in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*, New York, NY, USA, 2014, pp. 1975–1975.

[82] J. Acharya, I. Diakonikolas, C. Hegde, J. Z. Li, and L. Schmidt, Fast and near-optimal algorithms for approximating distributions by histograms, in *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'15)*, Melbourne, Australia, 2015, pp. 249–263.

[83] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. New York, NY, USA: Chapman & Hall, 1993.

[84] F. Olken and D. Rotem, Random sampling from database files: A survey, in *Proceedings of the 5th International Conference on Statistical and Scientific Database Management (SSDBM'1990)*, Charlotte, NC, USA, 1990, pp. 92–111.

[85] F. Olken, Random sampling from databases, PhD dissertation, University of California at Berkeley, Berkeley, CA, USA, 1993.

[86] P. Bruce and A. Bruce, *Practical Statistics for Data Scientists: 50 Essential Concepts*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2017.

[87] W. S. Cleveland and R. Hafen, Divide and recombine (d&r): Data science for large complex data, *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 7, no. 6, pp. 425–433, 2014.

[88] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri, Macrobase: Prioritizing attention in fast data, in *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD'17)*, Chicago, IL, USA, 2017, pp. 541–556.

[89] S. Chaudhuri, R. Motwani, and V. Narasayya, Random sampling for histogram construction: How much is enough?, *ACM SIGMOD Record*, vol. 27, no. 2, pp. 436–447, 1998.

[90] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, Random sampling techniques for space efficient online computation of order statistics of large datasets, in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, Philadelphia, PA, USA, 1999, pp. 251–262.

[91] M. Charikar, K. Chen, and M. Farach-Colton, Finding frequent items in data streams, in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, Málaga, Spain, 2002, pp. 693–703.

[92] S. Guha and A. McGregor, Space-efficient sampling, in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS 2007)*, San Juan, PUR, USA, 2007, pp. 169–176.

[93] F. Kuhn, T. Locher, and S. Schmid, Distributed computation of the mode, in *Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing (PODC'08)*, Toronto, Canada, 2008, pp. 15–24.

[94] A. Kirsch, M. Mitzenmacher, A. Pietracaprina, E. Upfal, and F. Vandin, A rigorous statistical approach for identifying significant itemsets, in *Proceedings of the IEEE International Conference on Data Mining (ICDM'08)*, Pisa, Italy, 2008, pp. 1–10.

[95] J. A. R. Rojas, M. Beth Kery, S. Rosenthal, and A. Dey, Sampling techniques to improve big data exploration, in *Proc. of 2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*, Phoenix, AZ, USA, 2017, pp. 26–35.

[96] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, Enterprise data analysis and visualization: An interview study, *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2917–2926, 2012.

[97] J. Lin and D. Ryaboy, Scaling big data mining infrastructure: The twitter experience, *SIGKDD Explorations*, vol. 14, no. 2, pp. 6–19, 2013.

[98] C. T. Fan, Development of sampling plans by using sequential (item by item) selection techniques and digital computers, *Publications of the American Statistical Association*, vol. 57, no. 298, pp. 387–402, 1962.

[99] P. J. Haas, Data-stream sampling: Basic techniques and results, in *Data Stream Management, Data-Centric Systems and Applications Book Series*. Berlin, Heidelberg, Germany: Springer, 2016, pp. 13–44.

[100] T. E. Oliphant, Scipy: Open source scientific tools for python, *Computing in Science and Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[101] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, The weka data mining software: An update, *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[102] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, *The Journal of Machine Learning Research*, vol. 12, no. 10, pp. 2825–2830, 2011.

[103] M. Al-Kateb and B. S. Lee, Stratified reservoir sampling over heterogeneous data streams, in *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM'10)*, Heidelberg, Germany, 2010, pp. 621–639.

[104] J. S. Vitter, Random sampling with a reservoir, *ACM Transactions on Mathematical Software*, vol. 11, no. 1, pp. 37–57, 1985.

[105] C. C. Aggarwal, On biased reservoir sampling in the presence of stream evolution, in *Proceedings of the 32nd International Conference on Very Large Databases (VLDB'06)*, Seoul, Korea, 2006, pp. 607–618.

[106] M. Dash and W. Ng, Efficient reservoir sampling for transactional data streams, in *Proc. of Sixth IEEE International Conference on Data Mining–Workshops (ICDMW'06)*, Hong Kong, China, 2006, pp. 662–666.

[107] V. Malbasa and S. Vucetic, A reservoir sampling

algorithm with adaptive estimation of conditional expectation, in *Proc. of International Joint Conference on Neural Networks*, Orlando, FL, USA, 2007, pp. 2200–2204.

[108] A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan, The big data Bootstrap, in *Proceedings of the 29th International Coference on International Conference on Machine Learning (ICML'12)*, Edinburgh, UK, 2012, pp. 1787–1794.

[109] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Boca Raton, FL, USA: CRC Press, 1984.

[110] L. Breiman, Bagging predictors, *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[111] M. I. Jordan, On statistics, computation and scalability, *Bernoulli*, vol. 19, no. 4, pp. 1378–1390, 2013.

[112] R. Genuer, J.-M. Poggi, C. Tuleau-Malot, and N. Villa-Vialaneix, Random forests for big data, *Big Data Research*, vol. 9, pp. 28–46, 2017.

[113] X. Meng, Scalable simple random sampling and stratified sampling, in *Proceedings of the 30th International Conference on International Conference on Machine Learning (ICML'13)*, Atlanta, GA, USA, 2013, pp. 531–539.

[114] P. Sanders, S. Lamm, L. Hübschle-Schneider, E. Schrade, and C. Dachsbacher, Efficient parallel random sampling–vectorized, cache-efficient, and online, *ACM Transactions on Mathematical Software*, vol. 44, no. 3, pp. 1–14, 2018.

[115] 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, *The Journal of Supercomputing*, vol. 74, no. 7, pp. 3415–3440, 2018.

[116] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, Synopses for massive data: Samples, histograms, wavelets, sketches, *Foundations Trends Databases*, vol. 4, no. 1, pp. 1–294, 2012.

[117] Q. Xu, T. Mytkowicz, and N. S. Kim, Approximate computing: A survey, *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.

[118] Y. Shi, X. Meng, F. Wang, and Y. Gan, HEDC: A histogram estimator for data in the cloud, in *Proceedings of the 4th International Workshop on Cloud Data Management (CloudDB'12)*, Maui, HI, USA, 2012, pp. 51–58.

[119] D. R. Krishnan, D. L. Quoc, P. Bhatotia, C. Fetzer, and R. Rodrigues, IncApprox: A data analytics system for incremental approximate computing, in *Proceedings of the 25th International Conference on World Wide Web (WWW'16)*, Montral, Canada, 2016, pp. 1133–1144.

[120] M. N. Garofalakis and P. B. Gibbon, Approximate query processing: Taming the terabytes, in *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, Roma, Italy, 2001, p. 725.

[121] N. Laptev, K. Zeng, and C. Zaniolo, Early accurate results for advanced analytics on MapReduce, *Proceedings of the VLDB Endowment*, vol. 5, no. 10, pp. 1028–1039, 2012.

[122] G. Hu, D. Zhang, S. Rigo, and T. D. Nguyen, Approximation with error bounds in spark, arXiv preprint arXiv: 1812.01823, 2018.

[123] Y. Park, J. Qing, X. Shen, and B. Mozafari, BlinkML: Approximate machine learning with probabilistic guarantees, in *Proc. of the 45th International Conference on Very Large Data Bases*, Los Angeles, CA, USA, 2018, pp. 1–18.

[124] X. Zhang, J. Wang, and J. Yin, Sapprox: Enabling efficient and accurate approximations on sub-datasets with distribution-aware online sampling, *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 109–120, 2016.

[125] S. Salloum, J. Z. Huang, Y. He, and X. Chen, An asymptotic ensemble learning framework for big data analysis, *IEEE Access*, vol. 7, no. 1, pp. 3675–3693, 2019.

[126] S. Salloum, J. Z. Huang, and Y. He, Exploring and cleaning big data with random sample data blocks, *Journal of Big Data*, vol. 6, no. 1, p. 45, 2019.

[127] Z. Wen, D. L. Quoc, P. Bhatotia, R. Chen, and M. Lee, ApproxIoT: Approximate analytics for edge computing, in *Proc. of 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 2018, pp. 411–421.

[128] M. Elteir, H. Lin, and W. Feng, Enhancing MapReduce via asynchronous data processing, in *Proc. of 2010 IEEE 16th International Conference on Parallel and Distributed Systems*, Shanghai, China, 2010, pp. 397–405.

Mohammad Sultan Mahmud is currently a PhD candidate at Shenzhen University, China. He received the master degree from King Mongkut's University of Technology North Bangkok, Thailand, in 2014, and the bachelor degree from BGC Trust University Bangladesh, Bangladesh, in 2008. Mr. Mahmud was awarded the Outstanding Doctoral Student of Shenzhen University in 2017 and Shenzhen Universiade International Scholarship in 2018. Also, he received Information Technology-King Mongkut's University of Technology North Bangkok scholarship for two years in 2012. His current research focuses on big data mining and distributed and parallel computing.

Joshua Z. Huang received the PhD degree from the Royal Institute of Technology, Sweden, in 1993. He is a distinguished professor of the College of Computer Science & Software Engineering at Shenzhen University. Also, he is the director of Big Data Institute and the deputy director of the National Engineering Laboratory for Big Data System Computing Technology. His main research interests include big data technology and applications. Prof. Huang has published over 200 research papers in conferences and journals. In 2006, he received the most influential paper award in the First Pacific-Asia Conference on Knowledge Discovery and Data

Mining. Prof. Huang 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.

Salman Salloum received the PhD degree from Shenzhen University, Shenzhen, China, in 2019, and the MS degree from Damascus University, Damascus, Syria, in 2013. He is currently an associate researcher with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. From 2007 to 2014, he had worked as an instructional designer and a project manager in ePedia-SY, a digital content company in Syria. He was also a tutor at Syrian Virtual University from 2012 to 2014. His current research is focused on cluster computing and approximate computing for big data analysis.

Tamer Z. Emara is currently a PhD candidate at Big Data Institute, Shenzhen University, China. In 2015, he got the MS degree from Mansoura University, Egypt. Also, he received the BS degree from Tanta University, Egypt, in 2005. He is now a lecturer at the Higher Institute of Engineering and Technology, Kafrelsheikh, Egypt. His main research interest is big data management. He is a member of IEEE and ACM.

Sadatdiynov Kuanishbay currently is a PhD candidate at Shenzhen University, China. He received the BS and the MS degrees from Tashkent University of Information Technologies, Uzbekistan, in 2012 and 2014, respectively. His research interests include edge computing, network architecture, and big data analysis.