A Two-Stage Data Processing Algorithm to Generate Random Sample Partitions for Big Data Analysis

Chenghao Wei, Salman Salloum, Tamer Z. Emara, Xiaoliang Zhang, Joshua Zhexue Huang(✉), and Yulin He

Big Data Institute, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China
{chenghao.wei,zx.huang}@szu.edu.cn

Abstract. To enable the individual data block files of a distributed big data set to be used as random samples for big data analysis, a two-stage data processing (TSDP) algorithm is proposed in this paper to convert a big data set into a random sample partition (RSP) representation which ensures that each individual data block in the RSP is a random sample of the big data, therefore, it can be used to estimate the statistical properties of the big data. The first stage of this algorithm is to sequentially chunk the big data set into non-overlapping subsets and distribute these subsets as data block files to the nodes of a cluster. The second stage is to take a random sample from each subset without replacement to form a new subset saved as an RSP data block file and the random sampling step is repeated until all data records in all subsets are used up and a new set of RSP data block files are created to form an RSP of the big data. It is formally proved that the expectation of the sample distribution function (*s.d.f.*) of each RSP data block equals to the *s.d.f.* of the big data set, therefore, each RSP data block is a random sample of the big data set. Implementation of the TSDP algorithm on Apache Spark and HDFS is presented. Performance evaluations on terabyte data sets show the efficiency of this algorithm in converting HDFS big data files into HDFS RSP big data files. We also show an example that uses only a small number of RSP data blocks to build ensemble models which perform better than the single model built from the entire data set.

Keywords: Big data analysis · Random sample partition · RSP HDFS · Apache Spark

The work of this paper was supported by National Natural Science Foundations of China (61503252 and 61473194), China Postdoctoral Science Foundation (2016T90799), Scientific Research Foundation of Shenzhen University for Newly-introduced Teachers (2018060) and Shenzhen-Hong Kong Technology Cooperation Foundation (SGLH20161209101100926).

1 Introduction

In the era of big data, large files of millions of objects with thousands of features are frequently encountered in many applications [1]. For example, millions of customers in a telecom company are required to be segmented into a large number of small groups for targeted marketing and product promotion [2]. Millions of people in a social network are analyzed to understand their opinions on different policies in different social groups [3]. Analyzing such an ever increasing big data is a challenging problem, even on computing clusters, especially when data volume goes beyond the available computing resources. In this regard, divide-and-conquer has become a common and necessary computing strategy to handle big data [4] by distributing both data and computation tasks to the nodes of clusters using distributed file systems (e.g., Hadoop distributed file system (HDFS)) [5] and big data processing frameworks (e.g., Hadoop's MapReduce [6,7], Apache Spark [8,9] and Microsoft R Server[1]). Nevertheless, all distributed data blocks of a data set must be loaded and processed when running data analysis algorithms with current big data frameworks. If the data set to be analyzed exceeds the memory of the cluster system, the algorithms will be inefficient or may not be able to analyze the data. Therefore, the size of memory is an important factor to the ability and performance of the current big data processing and analysis platforms. However, memory may never be enough considering the speed of the increase of big data in size. Can we have a technology with which analysis of big data will not be limited to the size of memory? The answer is affirmative.

If we only analyze some small distributed data blocks without considering the entire big data set and use the results of these blocks to estimate or infer the results of the entire big data set, then the above problem can be solved. In our previous empirical study [10], we found that it is possible if the distributed data blocks have similar probability distributions to the probability distribution of the big data set. We also found that if the records of a big data set are randomly organized, i.e., satisfying the *i.i.d.* condition, sequentially chunking the big data set into a set of data block files will make each data block a random sample of the big data, i.e., making the probability distribution of each data block similar to the probability distribution of the big data set. In statistical analysis, using random samples is a basic strategy to analyze a large data set or an unknown population [11]. The statistical properties of a big data set can be estimated and inferred from the results of random samples of the big data [12]. However, having a naturally randomized data set can only be taken as an exception rather than a rule in big data because it depends on the data generation process. Moreover, the data partitioning methods in current distributed file systems do not take the statistical properties of data into consideration. Consequently, data blocks produced by such methods may not necessarily be random samples of the big data set and using these data blocks as random samples to directly estimate statistics of the big data or build analytical models will lead to biased or incorrect

[1] https://www.microsoft.com/en-us/cloud-platform/r-server.

results. Therefore, a new big data representation model is required to support scalable, efficient and effective big data analysis.

Recently, we have proposed the random sample partition (RSP) data model for big data analysis. The RSP model represents a big data set as a set of non-overlapping subsets which forms a partition of the big data set. Each subset is saved as an RSP data block file that is used as a random sample of the big data set. In this way, analysis of a big data set approximately equals to analysis of one or more RSP data blocks which can be easily read in the nodes of a cluster and computed without the need of big memory. Therefore, the memory limitation on big data analysis is removed in the new RSP data representation model. However, current big data sets are usually saved as HDFS files on computing clusters. Efficient and scalable algorithms are required to convert existing big HDFS files into RSP representations in the scale of Terabyte to support big data analysis. Currently, such algorithms are not available.

In this paper, we propose a two-stage data processing (TSDP) algorithm that is used to convert a big HDFS data set into an RSP representation which ensures that each RSP data block is a random sample of the big data, therefore, it can be used to estimate the statistical properties of the big data. The first stage of this algorithm is to sequentially chunk a big data set into non-overlapping subsets and distribute these subsets as data block files on the nodes of a cluster. This operation is provided in HDFS. The second stage is to take a random sample from each subset without replacement to form a new subset saved as an RSP data block file. The random sampling step is repeated until all data records in all subsets are used up and a new set of RSP data block files are created to form an RSP of the big data. We formally prove that the expectation of the probability distribution of each RSP data block equals to the probability distribution of the big data set, therefore, each RSP data block is a random sample of the big data set.

We present the implementation of the TSDP algorithm on Apache Spark and HDFS and the performance evaluations of this algorithm on terabyte data sets. The experiments were conducted on a computing cluster with 29 nodes. The evaluation results have shown that the TSDP algorithm is efficient in converting HDFS big data files into HDFS RSP big data files. We also show an example that uses only few RSP data blocks to build ensemble models which perform better than the single model built from the entire data set.

The remainder of this paper is organized as follows. Section 2 presents preliminaries used in this research, including two basic definitions. The proposed TSDP algorithm is introduced in Sect. 3. Section 4 presents the TSDP implementation on Apache Spark. Experimental results are discussed in Sect. 5. Further discussions on the advantages of the RSP model for big data analysis and the TSDP algorithm are given in Sect. 6. Finally, the conclusions are given in Sect. 7.

2 Preliminaries

This section briefly reviews big data representations in Hadoop distributed file system (HDFS) and Apache Spark, and gives a formal data definition of the random sample partition (RSP) representation model.

2.1 Big Data Representations in HDFS and Spark

The current big data frameworks use divide-and-conquer as a general strategy to analyze big data on computing clusters. A big data set is chunked into small data blocks and distributed on the nodes of a computing cluster using distributed file systems such as Hadoop distributed file system (HDFS) [13]. To tackle a big data analysis task, the data blocks of a big data set are processed in parallel on the cluster. The intermediate results from the local data blocks processed on local nodes are integrated to produce the final result for the entire data set. To save a big data set as an HDFS file, HDFS operation sequentially chunks the big data set into small data blocks of fixed size (e.g., 64 MB or 128 MB) and distribute the data blocks randomly on the local nodes of the cluster. For data safety, 3 copies of the same data block are usually stored on three different nodes. Apache Spark provides processing-level distributed data abstractions as the resilient distributed data set (RDD) which is held in memory to facilitate the data processing and analysis. The RDDs APIs are provided to import HDFS files or other data files to Spark RDD, control RDD partitioning and manipulate RDD using a rich set of operators. An RDD can be partitioned and repartitioned using different partitioning methods such as hash partitioner and range partitioner.

However, these data partitioning methods, whether HDFS or Spark RDD, do not consider the statistical properties of the big data set. As a result, the data blocks in HDFS files and Spark RDDs cannot be used as random samples for statistical analysis of the big data set. Using these data blocks to estimate the big data tends to produce statistically biased results.

2.2 RSP Data Representation Model

To enable HDFS distributed data blocks to be used as random samples for estimation and analysis of the entire big data set, we propose to use RSP distributed data representation model to represent big data, which ensures that each data block in the RSP model is a random sample of the big data set. The main properties of the RSP model are given in the following two definitions.

Definition 1 (Partition of Data Set): Let $\mathbb{D} = \{\mathrm{x}_1, \mathrm{x}_2, \cdots, \mathrm{x}_N\}$ be a data set containing N objects. Let $\mathbb{T}$ be an operation which divides $\mathbb{D}$ into a family of subsets $\mathrm{T} = \{\mathrm{D}_1, \mathrm{D}_2, \cdots, \mathrm{D}_K\}$. T is called a *partition* of data set $\mathbb{D}$ if

(1) $\bigcup_{k=1}^{K} \mathrm{D}_k = \mathbb{D}$;

(2) $\mathrm{D}_i \cap \mathrm{D}_j = \emptyset$, when $i, j \in \{1, 2, \cdots, K\}$ and $i \neq j$.

Accordingly, $\mathbb{T}$ is called a partition operation on $\mathbb{D}$ and each $\mathrm{D}_k(k = 1, 2, \cdots, K)$ is called a data block of $\mathbb{D}$.

An HDFS file is a partition of data set $\mathbb{D}$ where data blocks $\{\mathrm{D}_1, \mathrm{D}_2, \cdots, \mathrm{D}_K\}$ are generated by sequentially cutting the big data. Usually, these data blocks in HDFS files do not have similar distribution properties as the big data, therefore, cannot be used in general as random samples for analysis of the big data set. However, the data blocks in the RSP as defined below can be used as random samples of the big data set.

Definition 2 (Random Sample Partition): Let $\mathbb{D} = \{\mathrm{x}_1, \mathrm{x}_2, \cdots, \mathrm{x}_N\}$ be a big data set which is a random sample of a population and assume $F(x)$ to be the sample distribution function (*s.d.f.*) of $\mathbb{D}$. Let $\mathbb{T}$ be a partition operation on $\mathbb{D}$ and $\mathrm{T} = \{\mathrm{D}_1, \mathrm{D}_2, \cdots, \mathrm{D}_K\}$ be a partition of data set $\mathbb{D}$ accordingly. T is called a *random sample partition* of $\mathbb{D}$ if

$$E[\tilde{F}_k(x)] = F(x) \quad \text{for each} \quad k = 1, 2, \cdots, K,$$

where $\tilde{F}_k(x)$ denotes the sample distribution function of $\mathrm{D_k}$ and $E[\tilde{F}_k(x)]$ denotes its expectation. Accordingly, each $\mathrm{D_k}$ is called an RSP block of $\mathbb{D}$ and $\mathbb{T}$ is called an RSP operation on $\mathbb{D}$.

3 Two-Stage Data Processing Algorithm

In this section, we present a two stage data processing algorithm for generating RSP from a big data set. Let $\mathbb{D} = \{\mathrm{x}_1, \mathrm{x}_2, \cdots, \mathrm{x}_N\}$ be a data set with N objects and M features. To generate RSP data blocks (i.e., random samples) from $\mathbb{D}$, if N is not big, we can easily use the following steps to convert $\mathbb{D}$ into Q RSP data blocks.

- Step 1: Generate N unique random integer numbers for N objects from a uniform distribution;
- Step 2: Sort N objects on the random numbers to reorganize $\mathbb{D}$;
- Step 3: Sequentially cut N reordered objects into Q small data blocks, each with N/Q objects.

Corollary 1. *Let* $\{\mathrm{D}_1, \mathrm{D}_2, \cdots, \mathrm{D}_Q\}$ *denote the above* Q *small blocks. Each* $\mathrm{D_k}$ *is an RSP data block of* $\mathbb{D}$*.*

Proof: Set $\mathrm{D}_k = \left\{\mathrm{x}_1^{(k)}, \mathrm{x}_2^{(k)}, \cdots, \mathrm{x}_{N_k}^{(k)}\right\}$, $k \in \{1, 2, \cdots, Q\}$ and N_k is the number of objects in D_k. It is obvious that

$$P\left\{\mathrm{D}_k = \left\{\mathrm{x}_{s_1}, \mathrm{x}_{s_2}, \cdots, \mathrm{x}_{s_{N_k}}\right\}\right\} = \frac{1}{C_N^{N_k}}, \tag{1}$$

where $x_{s_1}, x_{s_2}, \cdots, x_{s_{N_k}}$ are s_{N_k} objects selected arbitrarily from $\mathbb{D}$.

Assume $F(x)$ is the *s.d.f.* of $\mathbb{D}$. On one hand, for each real number $x \in R^1$, the number of samples whose values are not greater than x is $F(x) \cdot N$. On the other hand, for each data block $\mathrm{D_k}$ and object $x_i \in \mathbb{D}$, $P\{x_i \in \mathrm{D}_k\} = C_{N-1}^{N_k-1} \cdot \frac{1}{C_N^{N_k}} = \frac{N_k}{N}$. Thus the number of objects in D_k whose values are not greater than x is $F(x) \cdot N \cdot \frac{N_k}{N} = F(x) \cdot N_k$. We therefore obtain that the expectation of the *s.d.f.* of D_k is $\frac{1}{N_k} \cdot F(x) \cdot N_k = F(x)$. According to Definition 2, D_k is an RSP data block of $\mathbb{D}$.

When N is not large and $\mathbb{D}$ can be sorted in memory, the above algorithm works well. However, when N is large and $\mathbb{D}$ is big, e.g., in terabyte, if Q is also large, e.g., 100000 data blocks, the execution of this algorithm on the cluster becomes impractical, e.g., running extremely long time or bringing the cluster down due to many processes competing for resources. To deal with that situation, a general two-stage data-processing (TSDP) algorithm is designed for converting a big data set to a set of RSP data blocks below. Firstly, we give the following theorem before we present the TSDP algorithm.

Theorem 1. *Let $\mathbb{D}_1$ and $\mathbb{D}_2$ be two big data sets with N_1 and N_2 objects respectively. Assume that D_1 with n_1 objects is an RSP data block of $\mathbb{D}_1$ and D_2 with n_2 objects is an RSP data block of $\mathbb{D}_2$. Then, $\mathrm{D}_1 \bigcup \mathrm{D}_2$ is an RSP data block of $\mathbb{D}_1 \bigcup \mathbb{D}_2$ under the condition that $\frac{n_1}{n_2} = \frac{N_1}{N_2}$.*

Proof: Let $F_1(x)$ and $F_2(x)$ denote the *s.d.f.*s of $\mathbb{D}_1$ and $\mathbb{D}_2$ respectively. Assume that the *s.d.f.*s of D_1 and D_2 are $\tilde{F}_1(x)$ and $\tilde{F}_2(x)$, respectively. According to Definition 2, we have $E[\tilde{F}_1(x)] = F_1(x), E[\tilde{F}_2(x)] = F_2(x)$. For any real number x, the number of objects in $\mathrm{D}_1 \bigcup \mathrm{D}_2$ whose values are not greater than x is $n_1\tilde{F}_1(x) + n_2\tilde{F}_2(x)$. Therefore, the *s.d.f.* of $\mathrm{D}_1 \bigcup \mathrm{D}_2$ is:

$$\tilde{F}(x) = \frac{n_1\tilde{F}_1(x) + n_2\tilde{F}_2(x)}{n_1 + n_2}.$$

Similarly, the *s.d.f.* of $\mathbb{D}_1 \bigcup \mathbb{D}_2$ is:

$$F(x) = \frac{N_1F_1(x) + N_2F_2(x)}{N_1 + N_2}.$$

The expectation of $\tilde{F}(x)$ is

$$\begin{aligned} E[\tilde{F}(x)] &= E[\frac{n_1\tilde{F}_1(x) + n_2\tilde{F}_2(x)}{n_1 + n_2}] = \frac{n_1E[\tilde{F}_1(x)] + n_2E[\tilde{F}_2(x)]}{n_1 + n_2} \\ &= \frac{n_1F_1(x) + n_2F_2(x)}{n_1 + n_2} = \frac{N_1F_1(x) + N_2F_2(x)}{N_1 + N_2} \\ &= F(x). \end{aligned}$$

Therefore, $\mathrm{D}_1 \bigcup \mathrm{D}_2$ is an RSP data block of $\mathbb{D}_1 \bigcup \mathbb{D}_2$.

Remark: With a subtle modification, the proof of Corollary 1 can be extended to data in multiple dimensions and the proof of Theorem 1 can also be extended to the multiple dimensions and more than two data sets.

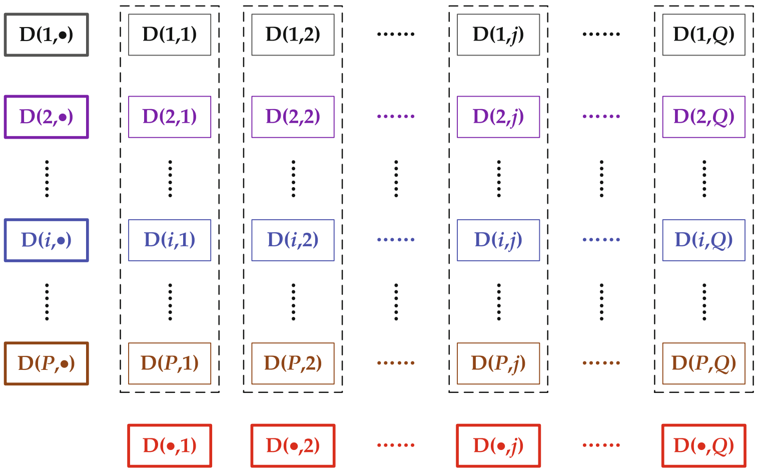


Fig. 1. Illustration of TSDP algorithm.

Let $\mathbb{D}$ be a data set of N objects and M features where N is too big for statistical analysis on a single machine. The following TSDF algorithm is designed to generate an RSP with Q RSP data blocks from $\mathbb{D}$ on a computing cluster.

Stage 1 (Data-Chunking): $\mathbb{D}$ is sequentially chunked into P data blocks $\{\mathrm{D}_i\}_{i=1}^{P}$ which form a partition of $\mathbb{D}$. HDFS provides functions to perform this operation. We call these blocks as HDFS data blocks which are not necessarily random samples of $\mathbb{D}$.

Stage 2 (Data-Randomization): Each RSP data block is built using random samples from P HDFS data blocks. This requires a distributed randomization operation where a slice of samples is selected randomly without replacement from each HDFS data block to form a new RSP data block. This operation is repeated Q times to produce Q RSP data blocks. The main steps in this stage are summarized as follows:

– Step 1: Randomizing D_i in parallel for $1 \leq i \leq P$;
– Step 2: Cutting each D_i in parallel into Q subsets $\{\mathrm{D}(i,j)\}_{j=1}^{Q}$ which form an RSP of D_i, for $1 \leq i \leq P$;
– Step 3: From $P \times Q$ subsets $\{\mathrm{D}(i,j)\}$ for $1 \leq i \leq P$ and $1 \leq j \leq Q$ which form a partition of data set $\mathbb{D}$, merge the subsets of $\{\mathrm{D}(i,j)\}_{i=1}^{P}$ for $1 \leq j \leq Q$ to form Q subsets $\{\mathrm{D}(\cdot,j)\}_{j=1}^{Q}$ which form an RSP of $\mathbb{D}$.

TSDP algorithm is illustrated in Fig. 1. The first stage generates the first level partition $\{\mathrm{D}(1,\cdot), \mathrm{D}(2,\cdot), \cdots, \mathrm{D}(P,\cdot)\}$ of $\mathbb{D}$. The second stage randomizes each subset $\mathrm{D}(i,\cdot)$ independently, generates the second level partitions $\{\mathrm{D}(i,1), \mathrm{D}(i,2), \ldots, \mathrm{D}(i,Q)\}$ for $1 \leq i \leq P$ and merges the column partitions into $\{\mathrm{D}(\cdot,1), \mathrm{D}(\cdot,2), \ldots, \mathrm{D}(\cdot,Q)\}$. By repeatedly applying Theorem 1, we can deduce that the new data block $\mathrm{D}(\cdot,j), (j = 1, \cdots, Q)$ is an RSP data block of $\mathbb{D}$.

4 TSDP Implementation on Apache Spark

In this section, we present an implementation of the TSDP algorithm on Apache Spark with HDFS. This implementation is called the *round-random partitioner* which is a hybrid between the range partitioner and the hash partitioner in Spark. Algorithm 1 shows the Pseudocode of this TSDP implementation in which the two stages are fulfilled as follows:

Algorithm 1. Round-Random Partitioner

```
Input:
- $\mathbb{D}_{HDFS}$: The HDFS file of a big data set $\mathbb{D}$;
- $P$: the number of RDD partitions in $\mathbb{D}_{HDFS}$;
- $Q$: the number of RSP data blocks in the RSP output HDFS file;
Method:
$\mathbb{D}_{RDD} \leftarrow$ read $\mathbb{D}_{HDFS}$; load HDFS file into an RDD
for all block $i$ in $\mathbb{D}_{RDD}$ do
    $K = Seq$ (1 to $n$)
    shuffle $K$;
end for
bind K with $\mathbb{D}_{RDD}$ partitions to generate $pairRDD(K, V)$;
repartition $pairRDD$ using HashPartitioner($Q$)
$\mathbb{D}_{HDFS-RSP}$ = values of $pairRDD$; ignore the random numbers in $\mathbb{D}_{HDFS-RSP}$
Output:
- $\mathbb{D}_{HDFS-RSP}$: the final RSP; Generate the RSP HDFS file.
```

Stage 1 (Data-Chunking): This stage is performed using HDFS. A given big data set $\mathbb{D}$ is converted to an HDFS file with P HDFS data blocks. The HDFS file of $\mathbb{D}$ is loaded to a Spark RDD using `SparkContext.textFile()` function. By default, each HDFS data block is mapped into an RDD partition and the Spark RDD consists of P RDD partitions. The number of RDD partitions to be generated can also be specified in the Spark data loading function. This step is basically a range partitioner which produces P HDFS data blocks[2].

Stage 2 (Data-Randomization): This stage randomizes all Spark partitions in parallel, randomly takes samples from each partition to form a set of Q RDD partitions and saves these RDD partitions as RSP data blocks in an HDFS file. This stage is completed with the following operations:

– Key generation: Generate an array of a sequence of integers K_i $(1 \leq i \leq P)$ for each RDD partition. The length of the sequence equals to the number of records in the RDD partition. This operation is carried out in parallel.

[2] Note: In Spark's terminology, an RDD is equal to a partition of the big data set. A partition is equal to a data block of the big data set. In this section, we use partition to indicate a data block of the big data set loaded to an Spark RDD in order to be consistent with Spark' terminology in this Spark implementation.

– Key Randomizing and Binding: Use Spark shuffle operation to randomize the integer sequences in the arrays Ks in parallel. Use the randomized integers in each array as random numbers and bind them to its corresponding RDD partition in a Spark partition pair named as pairRDD.
– Repartitioning: Repartition the pair RDD by hashing the random numbers in the pairRDD to generate Q new RDD partitions in the pairRDD.
– RSP HDFS file generation: Ignore the random numbers in the pairRDD and save the rest partitions as RSP data blocks in an HDFS file. This HDFS file is an RSP representation of the big data set $\mathbb{D}$.

Algorithm 2. Synthetic Data Generation

```
Input:
- $M$: number of features;
- $P$: number of data blocks;
- $N$: number of objects;
- $H$: number of clusters;
Method:
for all $h = 1$ to $H$ do in parallel
    $\mu(M)$ =Fill $(\mu_m)|\mu_m \in U\,[0, 10]$
    $\sigma(M)$ =Fill $(\sigma_m)|\sigma_m \in U\,(0, 1)$
    $D\_RDD \leftarrow GenerateRandomRDD(N/H, M, P, \mu, \sigma)$
    save $D\_RDD$ on HDFS
end for
Collect all files under one directory using HDFS APIs
```

5 Experiments and Results

In this section, we demonstrate the performance of Spark implementation of the TSDP algorithm on a computing cluster consisting of 2 name nodes and 27 data nodes, all running Centos 6.8. Each node has 16 cores (32 with Hyper-threading), 128 GB RAM and 1.1 TB disk storage. Several synthetic data sets were used in the experiments. We first present the method used to generate synthetic data sets. Then, we present the time performance of the algorithm in generating RSP data blocks from HDFS data sets of different sizes, and the time performance in generating different numbers of RSP data blocks from 1 terabyte HDFS data set in 1000 HDFS data blocks, and the time performance in generating the same number of RSP data blocks from 1 terabyte HDFS data set with different numbers of HDFS data blocks. After that, we illustrate the changes of probability distributions of HDFS data blocks and RSP data blocks. Finally, we use a real data example to show how the RSP data blocks are used to build ensemble models from only few RSP data blocks with accuracies equivalent to or even better than the model built from the entire data set.

5.1 Synthetic Data Generation

Different synthetic data sets with different numbers of objects N, different numbers of features M and different numbers of clusters H were generated and saved as HDFS files with different numbers of data blocks P. The clusters in the data sets have normal distributions with different means μ and standard deviations σ. Table 1 shows the characteristics of the synthetic data sets. The data generation algorithm is illustrated in Algorithm 2 which was implemented in Apache Spark. The data generation is performed in the following steps:

– Mean Generation: For each cluster, create a mean array for all features $\mu(M)$ and fill the array with values generated from the uniform distribution $\mathrm{U}\,(0, 10)$.
– Standard Deviation Generation: For each cluster, create a standard deviation array for all features $\sigma(M)$ and fill this array with values generated from the continuous uniform distribution $\mathrm{U}\,(0, 10)$.
– Cluster RDD Generation: Randomly generate a cluster RDD, D_RDD, for each cluster from the normal distribution with the following parameters, the number of objects $\frac{N}{H}$, the number of features M, the number of data blocks P, the array of features' means μ, and the array of features' standard deviations σ.
– Save and Collect: Save D_RDD on HDFS. After saving the RDDs for all clusters, they are collected under the same directory using HDFS APIs.

Table 1. Characteristics of synthetic data sets: N, M and H are the numbers of objects, features and clusters, respectively.

DS name	Total size	N	M	H	Associated tasks
DS001	≃100 GB	100,000,000	100	100	Classification
DS002	≃200 GB	200,000,000	100	100	Classification
DS003	≃300 GB	300,000,000	100	100	Classification
DS004	≃400 GB	400,000,000	100	100	Classification
DS005	≃500 GB	500,000,000	100	100	Classification
DS006	≃600 GB	600,000,000	100	100	Classification
DS007	≃700 GB	700,000,000	100	100	Classification
DS008	≃800 GB	800,000,000	100	100	Classification
DS009	≃900 GB	900,000,000	100	100	Classification
DS010	≃1 TB	1,000,000,000	100	100	Classification

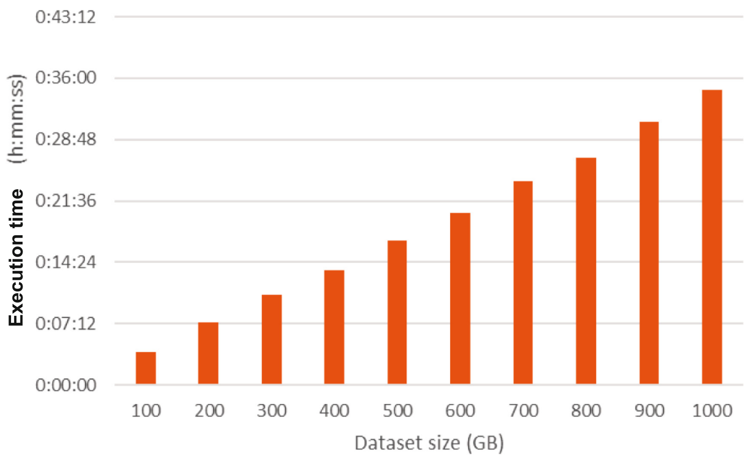


Fig. 2. Execution time for different data sets. The size of data set is changed from 100 GB to 1 TB, with 100 GB increase in each run.

5.2 Time Performance of TSDP

Three experiments were conducted with the synthetic data sets in Table 1 on the computing cluster. In the first experiment, all 10 synthetic data sets from 100 GB to 1 TB were used to convert HDFS data blocks to RSP data blocks of the these data sets. The size of both HDFS and RSP data blocks was fixed to 100 MB. The execution time for converting all HDFS data blocks to RSP data blocks for each data set by the TSDP algorithm on the computing cluster was recorded. Figure 2 shows the result of this experiment. We can see that when the size of data blocks is fixed, the time to generate RSP data blocks increases linearly to the size of the data set. We can observe that the execution time for 1 TB data set is about 35 min. Since we only need to transform each data set once, this time is quite acceptable in real applications. With this transformation, analysis on terabyte data becomes easier.

In the second experiment, only the data set in 1 TB was used. The number of HDFS data blocks was fixed to 1000. The number of RSP data blocks to be generated was increased from 10000 to 100000 with 10000 increase in each step. Figure 3(a) shows the execution time of ten runs on the computing cluster. We can see that the execution time increases linearly as the increase of the number of RSP data blocks to be generated when the number of HDFS data blocks was fixed to 10000. However, the time increase is not significant in each step, which indicates that the number of RSP data blocks is not an important factor to the execution time in this algorithm.

In the third experiment, only the data set in 1 TB was used. The number of RSP data blocks was fixed to 10000. The number of HDFS data blocks was changed from 10000 to 100000 with 10000 increase in each step. Figure 3(b) shows the execution time of ten runs on the computing cluster. Again, we can see the linear increase in execution time as increase in the number of HDFS data blocks in the HDFS file. However, the increase is more significant in each step. This indicates that the number of HDFS blocks in the HDFS file is an important factor to the execution time of this algorithm. We can see that converting 1 terabyte HDFS data set with 10000 HDFS blocks to 10000 RSP data blocks took about 30 min where converting the same data with 100000 HDFS blocks to 10000 RSP data blocks took more than 1 h. The reason is that the HDFS data blocks need to be randomized in this algorithm, which requires more memory resource to execute in parallel when the size and number of HDFS data block become large. These three experiments show that the TSDP algorithm is scalable to terabyte data, which facilitates the analysis of big data in terabyte size.

5.3 Distribution of Data Blocks

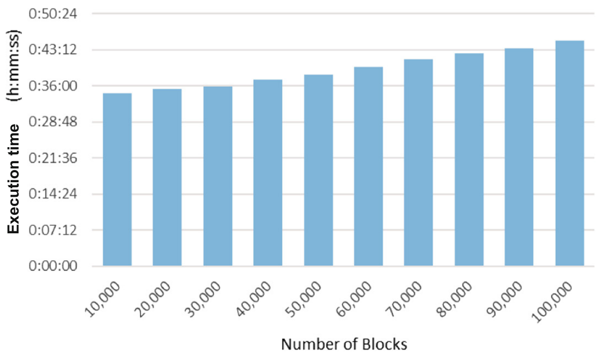
In this section, we show the probability distributions of features in HDFS data blocks and RSP data blocks and demonstrate that RSP data blocks of the same data set have similar probability distributions among themselves whereas the HDFS data blocks do not have similar probability distributions among themselves. Given a sequence of values of one feature from an HDFS data block, the probability density function (*p.d.f.*) is represented as a kernel density function which can be estimated by Parzen window method [14]. The density estimation algorithm is used to disperse the mass of the empirical *p.d.f.* over a regular grid with at least 512 points. The fast Fourier transform is used to convolve the approximation with a discredited version of the kernel. Gaussian basis functions are used to approximate univariate data. The rule-of-thumb is used to determine the bandwidth h of Parzen window [14] as

$$h = \left(\frac{4\sigma^5}{3n}\right)^{1/5} \approx 1.06\sigma n^{-1/5} \tag{2}$$

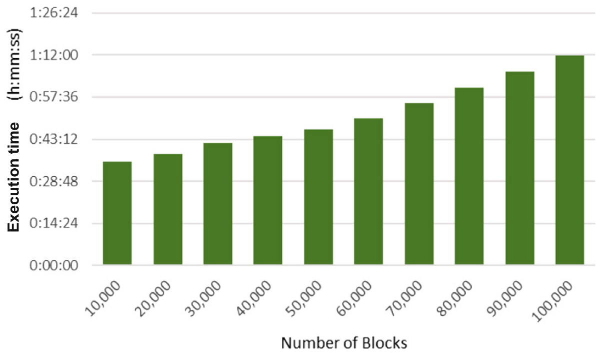
where σ is the standard deviation of the feature and n is the number of points. With this method, we plot the *p.d.f.* of any continuous feature in an HDFS data block and an RSP data block.

Figure 4(a) and (c) show the *p.d.f.* of two features from 6 randomly chosen HDFS data blocks in data set DS010. We can see that different HDFS data blocks have different probability distributions of two features. This indicates that these data blocks cannot be used to estimate and analyze the big data set because they are not representatives of the entire data set. To analyze the big data set, all HDFS data blocks must be taken into consideration, therefore, all data blocks have to be loaded to memory and compute. If the computing cluster has limited resource, then, the ability of analyzing big data will be limited.

Figure 4(b) and (d) show the *p.d.f.* of two features from 6 randomly chosen RSP data blocks in data set DS010. We can see that all RSP blocks have the same probability distribution on the same feature. Theorem 1 shows that the expectation of the probability distribution of each RSP data block is the probability distribution of the entire data set. Therefore, these RSP data blocks



(a) The number of RSP data blocks Q is changed from 10000 to 100000 blocks, while the number of HDFS data blocks P is fixed to 10000.



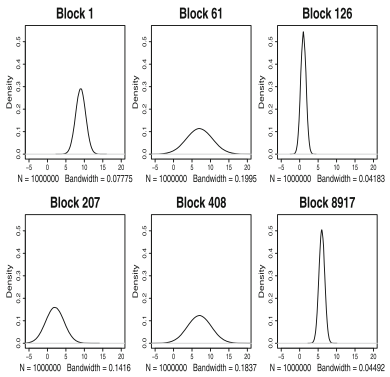
(b) The number of HDFS data blocks P is changed from 10000 to 100000 blocks, while the number of RSP data blocks Q is fixed to 10000.

Fig. 3. Execution time against the numbers of both HDFS and RSP data blocks in 1 terabyte data set.

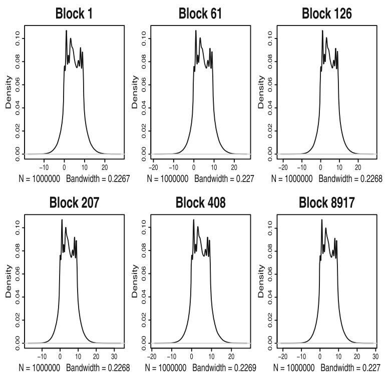
are good representatives of the entire data set and can be used to estimate and analyze the big data set. This is the rational to use RSP data blocks as random samples to analyze the big data set. After converting an HDFS big data set to RSP data blocks, analyzing the big data set is transferred to analysis of RSP data blocks. The big data set no longer needs to compute directly. Therefore, RSP representation significantly facilitates the analysis of big data and enables terabyte data sets to be easily analyzed.

5.4 RSP Block-Based Ensemble Classification

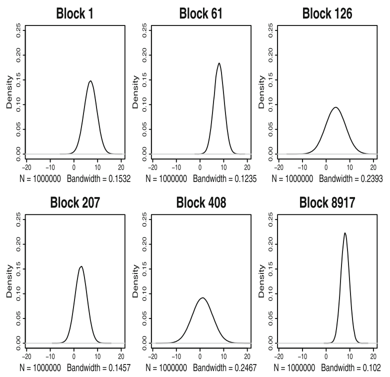
In this section, we use a real data example to show the use of few RSP data blocks to build ensemble classification models which perform equally good or even better than the model built from the entire data set. In this experiment, we used the asymptotic ensemble framework (called Alpha framework) [10] to build ensemble models from RSP data blocks.



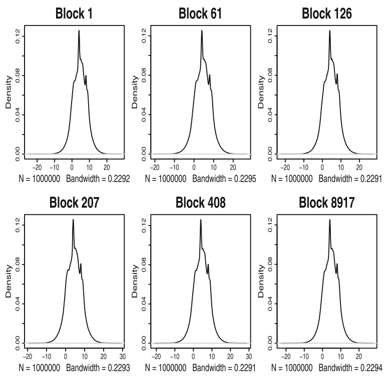
(a) *p.d.f.* without TSDP method on feature 1



(b) *p.d.f.* with TSDP method on feature 1



(c) *p.d.f.* without TSDP method on feature 2



(d) *p.d.f.* with TSDP method on feature 2

Fig. 4. *p.d.f.* comparison between HDFS data blocks and RSP data blocks in 1 TB synthetic data.

Alpha framework is illustrated in Fig. 5. A big data set $\mathbb{D}$ is first converted into RSP data blocks using the TSDP algorithm. Then, a subset of RSP data blocks is randomly selected without replacement, e.g., D_5, D_{120}, D_{506} and D_{890} in Fig. 5. After that, a base model is built from each of these data blocks, e.g., four classifiers $\pi_1, \pi_2, \pi_3, \pi_4$ built in parallel from 4 selected RSP data blocks. The next operation is to update the ensemble model Π with the newly built classifiers and evaluate Π. If Π satisfies the termination condition(s), then output Π, otherwise, go back to RSP representation and randomly select a new batch of RSP data blocks to build a second batch of new classifiers, update the ensemble model Π with the second batch of classifiers and evaluate it again. This process continues until a satisfactory ensemble model Π is obtained or all RSP data blocks are used up.

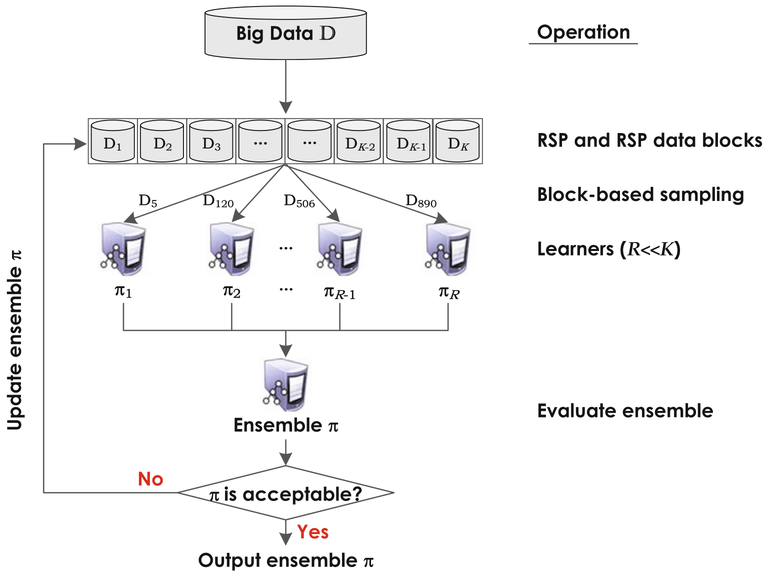
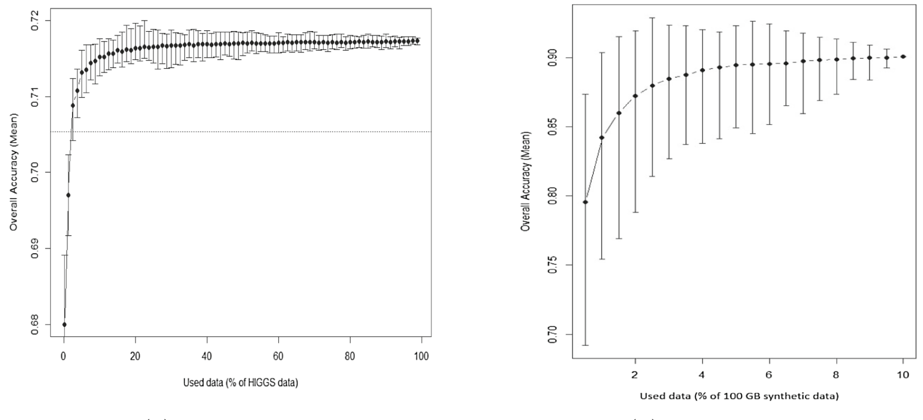


Fig. 5. Alpha framework: an asymptotic ensemble framework to build ensemble models from RSP data blocks.

The real data HIGGS with $N = 11,000,000$ objects and $M = 28$ features was used in this experiment. 400 RSP data blocks were generated with the TSDP algorithm. Figure 6(a) shows the accuracy of the ensemble models after each batch of data blocks. We can see that there is no significant change after using about 15% of the data and the accuracy value converges to a value which is approximately the same as the accuracy of a single model built from the whole data set (the dotted line in the figure). In addition, we also used the 100 GB synthetic data set DS001 to test the performance of the ensemble models from RSP data blocks. We found that only 10% of this data is enough to build an ensemble model with 90% accuracy as shown in Fig. 6(b). In such cases, we do not need to continue building further models from the remaining data blocks.

6 Discussions

Two computational costs are significant in cluster computing: one is the cost of communications among the computing nodes and the other is the cost of i/o operations on read/write data from/to the disks of local nodes. The two costs limit the ability of cluster computing on analysis of big data with complex algorithms. The in-memory computing technology alleviates the i/o cost but the communication cost can still be significant in analysis of big data. One solution is to use less sample data to estimate the big data. However, conducting random sampling on distributed big data sets is still very expensive because of the i/o and communication costs in cluster computing. RSP big data presentation model on cluster computing offers a fresh new solution to analysis of a big data set if the big data set is converted to a set of distributed RSP data blocks. In this new approach, analysis of big data becomes analysis of few randomly selected RSP data blocks from the RSP data representation. Three advantages can be summarized as follows:



(a) HIGGS data (b) Synthetic data

Fig. 6. Ensemble classification using RSP data blocks. Each point represents the overall ensemble accuracy calculated after each batch of blocks (averaged from 100 runs of the ensemble process).

– RSP block based analysis can estimate the statistical results of the big data set without computing the entire data set. Therefore, after a big data set is converted to a set of RSP data blocks, the size of the big data is no longer a barrier in big data analysis. This increases the ability of cluster computing when data volume exceeds the available resources.
– Since each RSP data block is processed independently on a local node with the same algorithm, sequential algorithms can be used on RSP data blocks without need of parallelization in the asymptotic ensemble learning framework. This approach makes many existing sequential algorithms useful in analysis of big data.
– Theorem 1 provides a theoretical foundation for analysis of a big data set distributed in multiple computing clusters. For example, if a big data set A is divided into three parts, (A_1, A_2, A_3) stored in three computing clusters (C_1, C_2, C_3), respectively. The TSDP algorithm can be used to convert each A_i to RSP blocks on C_i for $1 \leq i \leq 3$. To analyze A, we can randomly select

the same number of RSP data blocks from each computing cluster and merge the three sets of RSP data blocks into one set of RSP data blocks for A where each new RSP block is formed by merging three RSP blocks, each from one computing cluster. According to Theorem 1, the new blocks are RSP data blocks of A and can be used to estimate the statistical properties of A or build ensemble models for A. This new approach provides an opportunity to analyze big data sets cross data centers. However, detail technology needs to be further developed to facilitate analysis of big data sets cross multiple data centers.

7 Conclusions and Future Work

In this paper, we have proposed a two-stage data processing (TSDP) algorithm to generate the random sample partitions (RSPs) from HDFS big data sets for big data analysis. This algorithm is an enabling technology for RSP data block based analysis of big data sets, which does not need to compute the entire big data set. We have presented an implementation of the TSDP algorithm on Apache Spark and demonstrated the time performance of this algorithm on 1 terabyte data. The experiment results show that conversion of an HDFS data set in 1 terabyte was completed in less one hour which is well acceptable in real applications. We have also shown a real data example to demonstrate that the ensemble models built using few RSP data blocks performed better than the model built from the entire data set. Our future work is to extend the current algorithm to work on bigger data sets, e.g., 10 terabytes. We will also develop a distributed big data management system on computing clusters based on the RSP representation model.

References

1. Fan, J., Fang, H., Han, L.: Challenges of big data analysis. Nat. Sci. Rev. **1**(2), 293–314 (2014)
2. Zhao, J., Zhang, W., Liu, Y.: Improved K-means cluster algorithm in telecommunications enterprises customer segmentation. In: IEEE International Conference on Information Theory and Information Security, pp. 167–169 (2010)
3. Michael, B.: Uncovering online political communities of Belgian MPs through social network clustering analysis. In Proceedings of the ACM 2015 2nd International Conference on Electronic Governance and Open Society, pp. 150–163 (2015)
4. Ahmad, A., Paul, A., Rathore, M.M.: An efficient divide-and-conquer approach for big data analytics in machine-to-machine communication. Neurocomputing **174**(86), 439–453 (2016)
5. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: IEEE 26th Symposium Mass Storage Systems and Technologies, pp. 1–10 (2010)
6. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)

7. Elteir, M., Lin, H., Feng, W.C.: Enhancing mapreduce via asynchronous data processing. In: IEEE International Conference on Parallel and Distributed Systems, pp. 397–405 (2010)
8. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: HotCloud 2010, p. 10 (2010)
9. Salloum, S., Dautov, R., Chen, X., Peng, P.X., Huang, J.Z.: Big data analytics on apache spark. Int. J. Data Sci. Anal. **1**(3–4), 145–164 (2016)
10. Salloum, S., Huang, J.Z., He, Y.L.: Empirical analysis of asymptotic ensemble learning for big data. In: Proceedings of the 2016 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, pp. 8–17 (2016)
11. Cormode, G., Duffield, N.: Sampling for big data: a tutorial. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, p. 1975 (2014)
12. Garcia, D., Lubiano, M.A., Alonso, M.C.: Estimating the expected value of fuzzy random variables in the stratified random sampling from finite populations. Inf. Sci. **138**(4), 165–184 (2001)
13. Leo, S., Zanetti, G.: Pydoop: a python mapreduce and HDFS API for hadoop. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 819–825 (2010)
14. Sheather, S.J., Jones, M.C.: A reliable data-based bandwidth selection method for kernel density estimation. J. Roy. Stat. Soc. **53**(3), 683–690 (1991)