Science of Computer Programming 184 (2019) 102301

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Original software publication

RRPlib: A spark library for representing HDFS blocks as a set of random sample data blocks

Tamer Z. Emara [a,b,c,*], Joshua Zhexue Huang [a,b]

[a] Big Data Institute, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, Guangdong, China
[b] National Engineering Laboratory for Big Data System Computing Technology, Shenzhen University, Shenzhen 518060, Guangdong, China
[c] Higher Institute of Engineering and Technology Kafrelsheikh, Kafrelsheikh, Egypt

ARTICLE INFO

Article history:
Received 18 April 2019
Received in revised form 3 July 2019
Accepted 15 August 2019
Available online 18 September 2019

Keywords:
HDFS
Random sample
Data partitioning
Distributed systems

ABSTRACT

Analyzing big data is a challenging problem in cluster computing systems especially when the data volume goes beyond the available computing resources. Sampling is the favored solution for such problems. It summarizes or reduces the amount of data, taking into consideration the statistical characteristics of data distribution. However, the traditional method to sample the massive data by drawing record-by-record is a computationally expensive process because a full scan of the whole data is needed to be performed. While if the massive data is partitioned into a set of data blocks with each block is a random sample data block, the processing time for selecting some blocks as a sample (or different samples) is computationally cheaper. The main purpose of the software described in this paper is to represent the HDFS blocks as a set of random sample data blocks which also stored in HDFS. Our empirical results show that the performance of the partitioning operation is acceptable in the real application especially this operation is only performed once, thereby analysis on terabyte data becomes more natural.

* Corresponding author at: Big Data Institute, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, Guangdong, China.
E-mail addresses: tamer@szu.edu.cn (T.Z. Emara), zx.huang@szu.edu.cn (J.Z. Huang).

https://doi.org/10.1016/j.scico.2019.102301
0167-6423/

Software metadata

(executable) Software metadata description	
Current software version	1.0
Permanent link to executables of this version	https://github.com/ScienceofComputerProgramming/SCICO-D-19-00096
Legal Software License	Apache License 2.0
Computing platform/Operating System	CentOS version 6.8, Apache Hadoop version 2.6.0, and Apache Spark version 2.0.0
Installation requirements & dependencies	all dependencies in a pom.xml Maven file
If available, link to user manual – if formally published include a reference to the publication in the reference list	https://temara.github.io/RRPlib/
Support email for questions	amer@szu.edu.cn

Code metadata

Code metadata description	
Current code version	v1
Permanent link to code/repository used of this code version	https://github.com/ScienceofComputerProgramming/SCICO-D-19-00096
Legal Code License	Apache License 2.0
Code versioning system used	git
Software code languages, tools, and services used	scala
Compilation requirements, operating environments & dependencies	Maven project, can be compiled as a library (jar)
If available Link to developer documentation/manual	https://temara.github.io/RRPlib/
Support email for questions	amer@szu.edu.cn

1. Introduction

With the continuous rising of cloud computing and Internet of Things technologies, a massive data is being continuously generated in all fields of life on a daily basis. This amount of data is increasing at a rapid pace, which leads to challenges in storage, analysis, and various processing tasks. Hadoop [1] and Spark [2,3] are the most popular cluster-based applications emerged for distributed and parallel processing of big data. The basic idea of these systems is to cut the big data file sequentially into small files (mostly called blocks) and distribute them across the nodes of the cluster for parallel processing. However, this partitioning method does not consider the statistical characteristics of data distribution in the produced blocks. Thus, the general strategy of the distributed and parallel processing frameworks is to go through the entire data file to process it. Since the capacity of current technologies highly depends on in-memory computations, the analysis of the whole data file becomes a challenging matter, especially when it requires more than the available resources.

Sampling is the favored solution for such problems. It summarizes or reduces the amount of data, taking into consideration the statistical characteristics of data distribution. Recently, different resampling-based techniques have been proposed to tackle such a problem, for instance, bootstrap [4], subsampling [5], and a bag of little bootstrap [6]. The prevailing idea between these methods is that they draw their samples record-by-record. The computational complexity of such a process is expensive because a full scan of the entire data is needed to be performed. While if the massive data is partitioned into a set of data blocks with each block is a random sample data block, the processing time for selecting some blocks as a sample (or different samples) is computationally cheaper. Recently, round random partitioning algorithm (RRP) [7] has been proposed to represent the HDFS blocks as a set of random sample data blocks which also stored in HDFS.

The primary purpose of this work is to introduce the design and the implementation of RRPlib. It mainly has three components data generator, RRP, and massive-RRP. The purpose of the data generator component is to generate synthetic datasets for classification and regression. RRP component is an implementation example for the round-random partitioner in [7] and also the same for the massive-RRP component.[1]

2. Software description

This Library is a valuable tool to make the analysis process of the massive data an attainable operation. The code is written in Scala and available on Github repository. This library enables users to access the data through an application programming interface (API) or a command-line console.

2.1. Data generator

The simulated data is used to evaluate the performance of different machine learning algorithms running on large-scale datasets. Mainly, for this reason, the data generator is designed to generate synthetic datasets for various algorithms. Precisely, we consider the two most popular learning tasks, namely the classification and regression.

For classification, the data is generated with the form $Z_i = (X_i, Y_k)$, IID for $i = 1, ..., N$, $\forall Y_k \in (Y_1, ..., Y_K) : X_{i,m} = N(\mu_{k,m}, \sigma_{k,m})$, where $m = 1, ..., M$, $\mu_{k,m} \in U(0, 10)$ and $\sigma_{k,m} \in U(0, 10)$. While for the regression tasks, the data has the

[1] We are not explaining the analytical proof of these components. Interested readers may refer to [7–9].

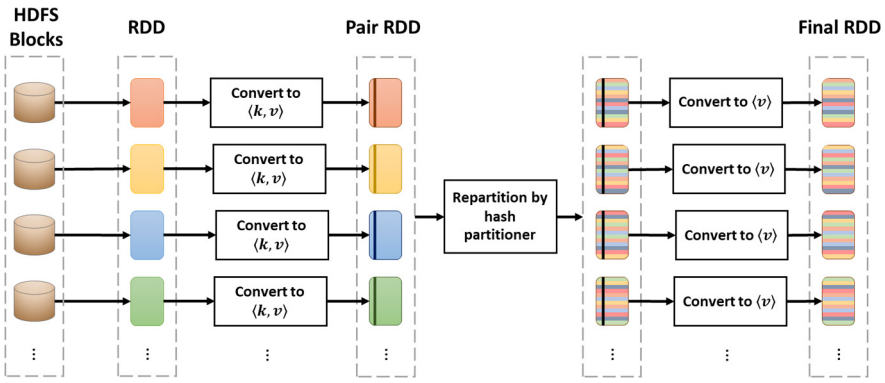


Fig. 1. The implementation steps of RRP using Apache Spark.

form $Z_i = (X_i, Y_i)$ which is independently and identically distributed (iid) for $i = 1, ..., N$. The library is designed to give the user the ability to pass a function which represents the label feature formula. In future work, we plan to enable data generation for various probability distributions and different tasks.

2.2. RRP: round-random partitioner

RRP [7] is proposed to partition a big file to be as a set of non-overlapping blocks, where each block can be considered as a random sample of the entire file. It is implemented based on Apache Spark framework [2,3]. Fig. 1 shows the main steps of RRP implementation which are summarized as follows:

- For a big file **D** stored in HDFS system, and has P blocks. Using `SparkContext.textFile(...)` operation, import the file into an RDD and maps each HDFS block as an RDD partition.
- For each RDD partition, each record value is mapped to pair value $< k, v >$, where $k \in \{1, 2, ..., n\}$ is selected randomly without replacement, and n is the number of the current partition records. Indeed, this operation is applied to the RDD partitions through the transformation `RDD.mapPartitions(...)`.
- After mapping the RDD values to pair values, the resulted RDD can be repartitioned using `HashPartitioner(Q)`. In fact, the hash partitioner reassigns each record to a new RDD partition index which is the result of $k \, mod \, Q$.
- Get the value only from the produced RDD to represent the final RDD.

Finally, we have to note that the type of this operation is a *transformation operation* which has lazy evaluation. This kind of operation needs an *action operation* to perform such as `RDD.saveAsTextFile(...)`

2.3. Massive-RRP: round-random partitioner for massive data

The massive-RRP component is proposed as an alternative solution in case that the data exceeds the limited resources, that makes it challenging to apply RRP algorithm. The basic idea is to divide the data into some groups of blocks allowing to apply RRP algorithm in the limited resources range. In the massive-RRP implementation, we assume the blocks are stored in separate files, and the big file is physically the folder which contains these blocks' files. This operation can be done by saving the file using the Spark's API, `RDD.saveAsTextFile(...)`. Fig. 2 demonstrates the steps of the massive-RRP algorithm, which can be summarized as follows:

- For a big file **D** is stored as a folder contains P files that represent the blocks.

Stage 1

- Distribute the files into d folders. The user is responsible for determining the number d with taking into consideration the currently available resources that allow being processed. Each folder consists of $\frac{P}{d}$ blocks, which are selected randomly with equal probabilities and without replacement.
- Apply RRP on each folder to produce a new folder contains a number of $\frac{P}{d}$ files which represent the random sample data block.

Stage 2

- Redistribute the resulted files from the last step to form d other folders. Each folder has $\frac{P}{d}$ files randomly selected from the resulted folders without replacement as demonstrated in step 3 in Fig. 2.

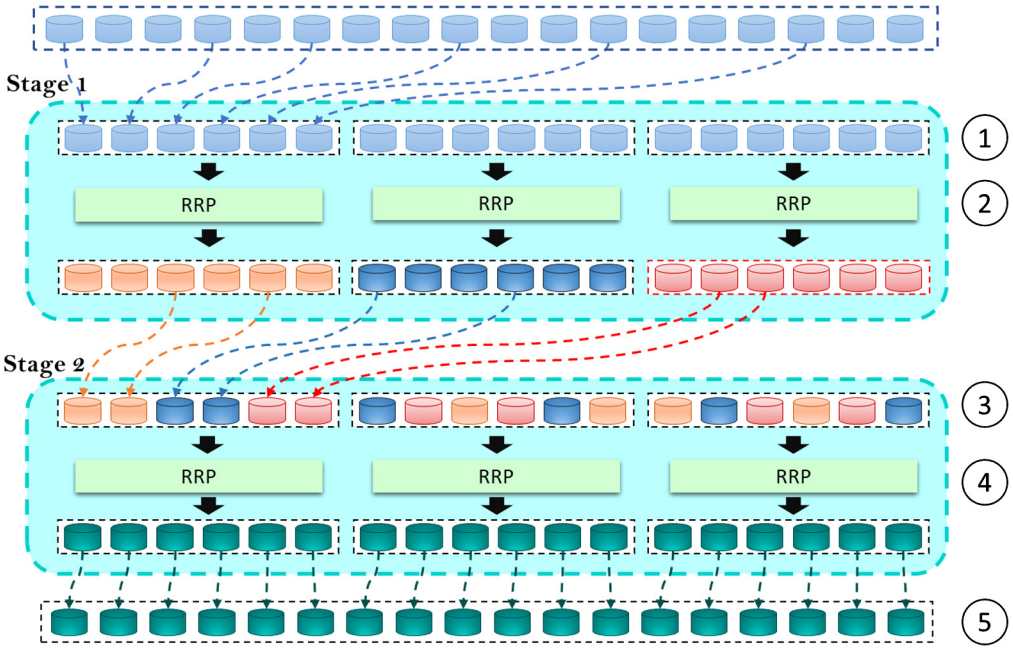


Fig. 2. Illustration of massive-RRP: it consists of two stages of randomization to generate the random sample data blocks.

Table 1
Characteristics of synthetic datasets.

DS name	Total size	N	M	P
DS001	≃ 100 GB	100,000,000	100	1000
DS002	≃ 200 GB	200,000,000	100	2000
DS003	≃ 300 GB	300,000,000	100	3000
DS004	≃ 400 GB	400,000,000	100	4000
DS005	≃ 500 GB	500,000,000	100	5000
DS006	≃ 600 GB	600,000,000	100	6000
DS007	≃ 700 GB	700,000,000	100	7000
DS008	≃ 800 GB	800,000,000	100	8000
DS009	≃ 900 GB	900,000,000	100	9000
DS010	≃ 1 TB	1,000,000,000	100	10000
DS011	≃ 10 TB	10,000,000,000	100	100000

- Apply RRP on each folder to produce a new folder contains $\frac{Q}{d}$ files which represent the random sample data block.
- Finally, collect all files under one folder directory to form one dataset which has Q files; each file is a random sample of the entire data.

3. Illustrative examples

To evaluate the usefulness of the RRPlib library, we conducted several experiments as examples to show how the library can enhance and speed up the performance of data processing and analysis. The experiments were conducted on a cluster consisting of 5 nodes. Each node has 12 cores (24 with Hyper-threading), 128 GB RAM and 12.5 TB disk storage. CentOS version 6.8, Apache Hadoop version 2.6.0 and Apache Spark version 2.0.0, are installed on this cluster.

We generated different synthetic datasets with varying numbers of records N, and numbers of features M. Table 1 shows the characteristics of the generated datasets. The generation process can be done through the command-line console. For example, in order to generate a dataset named $DS001$ with the following parameters: number of features $(M) = 100$, number of classes $(K) = 100$, number of records $(N) = 100,000,000$, and number of blocks $P = 1000$, we apply the following command line,

```
$ spark-submit --class szu.bdi.generators.GenerateDataSet \
  RRPlib.jar DS001 100000000 100 100 1000
```

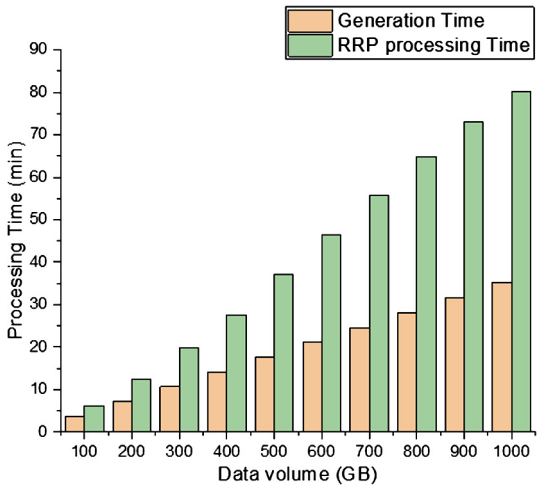


Fig. 3. Processing time of the generation and RRP operations.

We applied the same command with different parameters for all datasets listed in Table 1.

After generating the datasets, we repartitioned them using RRP through the following command:

```
$ spark-submit --class szu.bdi.apps.partitionerApp RRPlib.jar \
  <iDS> <oDS> <Q>
```

Where `iDS` is the path of the input dataset, `oDS` is the path of the output dataset, and `Q` is the partitions number of the output dataset.

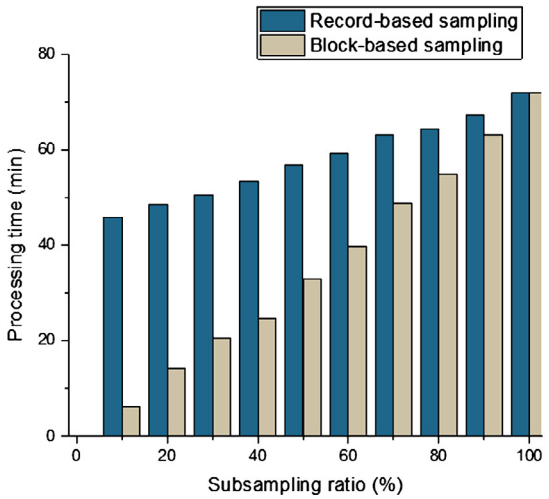
As an example, to generate the dataset `DS001P` from `DS001` with the same number of partitions `1000`, we executed the following command:

```
$ spark-submit --class szu.bdi.apps.partitionerApp RRPlib.jar \
  DS001 DS001P 1000
```

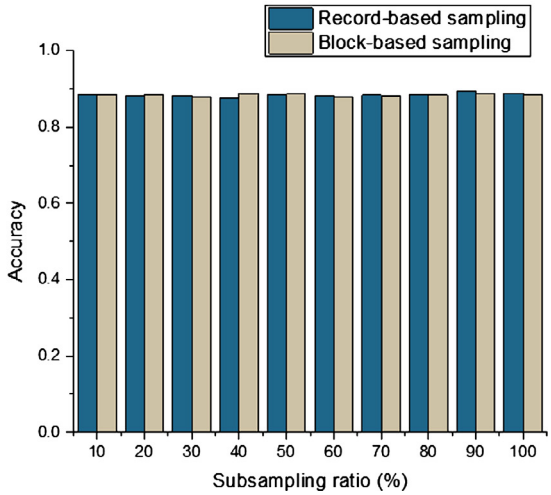
The user also has the ability to import the jar file and use the library's API, `RRP(...).partitionByRRP(...)`. A complete example demonstrates how to use it is as follows,

```
import org.apache.spark.sql.SparkSession
import szu.bdi.partitioner._

val spark = SparkSession.builder.master("yarn").getOrCreate()

val sourcePath: String = "DS001"
val finalPath: String = "DS001P"
val numPartReq = 1000

val sc = spark.sparkContext
val datasetRDD = sc.textFile(sourcePath)

val finalRDD = RRP(datasetRDD).partitionByRRP(numPartReq)
finalRDD.saveAsTextFile(finalPath)
```

During the previous experiments, the execution time for the generation and partitioning operations are recorded. The recorded values of the execution time are plotted in Fig. 3. It is evident that the execution time is quite acceptable in real applications since it is needed to transform each dataset once.



(a) Processing time versus the ratio of subsamples to generated data for classification tasks



(b) Accuracy versus the ratio of subsamples to generated data for classification tasks

Fig. 4. Classification results for random forest algorithm applied on subsamples selected from *DS010*.

To show the importance of this library, we tested the resulted blocks through evaluating the performance of classifying the dataset `DS010` using random forest algorithm regarding the accuracy and the processing time on subsamples of the data. These subsamples are drawn based on two methods, namely, block-based sampling (BBS) and record-based sampling (RBS). In block-based sampling, the blocks are selected directly as subsamples, while in record-based sampling, the samples are drawn record-by-record.

Fig. 4 shows that the accuracy of both methods is almost the same, while the processing time of classifying a small ratio of data using BBS is significantly shorter than using RBS method. Despite the consumed time to classify 100% of the data is the same for both methods, it is evident that the small ratio of the data is enough to estimate a model or a task.

4. Conclusions

The RRPlib library introduced in this paper is a ready-to-use tool to represent the big data as a set of random sample data blocks. The empirical results show that the execution time for partitioning operation is acceptable in the real application especially this operation is only performed once, thereby analysis on terabyte data becomes more natural. The significant impact of this system can be achieved by analyzing distributed data on multiple data-centers, and block-based sampling in resampling-based methods for statistical inference. Analyzing distributed data on multiple data-centers is a complicated process, especially with massive data. This library introduces a solution to facilitate the analysis process by selecting some blocks from different data centers and merging these blocks to produce a new randomized block. This new block can be considered as a random sample of the whole distributed data. Furthermore, selecting blocks as subsamples in different resampling-based methods can enhance and speed up the performance of data processing and analysis.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under grant No. 61972261.

References

[1] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST '26, IEEE Computer Society, Incline Village, NV, USA, 2010, pp. 1–10. URL http://gen.lib.rus.ec/scimag/index.php?s=10.1109/MSST.2010.5496972.

[2] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, in: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 1–7. URL http://dl.acm.org/citation.cfm?id=1863103.1863113.

[3] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, Apache spark: a unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65, https://doi.org/10.1145/2934664. URL http://doi.acm.org/10.1145/2934664.

[4] B. Efron, Bootstrap methods: another look at the jackknife, Ann. Stat. 7 (1) (1979) 1–26, https://doi.org/10.1214/aos/1176344552, arXiv:1306.3979v1. URL http://projecteuclid.org/euclid.aos/1176344552.

[5] D.N. Politis, J.P. Romano, M. Wolf, Subsampling, Springer-Verlag, New York, 1999.

[6] A. Kleiner, A. Talwalkar, P. Sarkar, M.I. Jordan, A scalable bootstrap for massive data, J. R. Stat. Soc., Ser. B, Stat. Methodol. 76 (4) (2014) 795–816, https://doi.org/10.1111/rssb.12050, arXiv:1112.5016.

[7] T.Z. Emara, J.Z. Huang, A distributed data management system to support large-scale data analysis, J. Syst. Softw. 148 (2019) 105–115, https://doi.org/10.1016/j.jss.2018.11.007. URL http://www.sciencedirect.com/science/article/pii/S0164121218302437.
[8] C. Wei, S. Salloum, T.Z. Emara, X. Zhang, J.Z. Huang, Y. He, A two-stage data processing algorithm to generate random sample partitions for big data analysis, in: M. Luo, L.-J. Zhang (Eds.), Cloud Computing – CLOUD 2018, Springer International Publishing, Cham, 2018, pp. 347–364.
[9] S. Salloum, Y. He, J. Zhexue Huang, X. Zhang, T.Z. Emara, A random sample partition data model for big data analysis, arXiv:1712.04146.