IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 3, Ver. III (May - June 2017), PP 25-28
www.iosrjournals.org

Distributed Clustering for Big Data with MapReduce

Ch.Radhika, D.V.Lalita Parameswari
G.Narayanamma Institute of Technology and Science, Hyderabad, India

Abstract: *Every day, a large volume of data is generated by multiple sources, social networks, mobile devices, etc. This variety of data sources produce a heterogeneous data, which are engendered in high frequency. One of the techniques allowing to a better use and exploit this kind of complex data is clustering. Finding a compromise between performance and speed response time present a major challenge to classify this data. This paper aims to propose efficient Distributed clustering algorithms to handle big data with MapReduce which is an improved version of single machine clustering.*
Index terms: *Big Data, Distributed, Clustering, MapReduce*

I. Introduction

MapReduce is one of the most efficient big data solutions, which enables to process a massive volume of data in parallel with many low-end computing nodes. This programming paradigm is a scalable and fault-tolerant data processing tool. It has gained significant momentum from industry and academia by virtue of its simplicity, scalability, and fault tolerance [3].

MapReduce model hides details of parallel execution, which allows users to focus only on data processing strategies. This model consists of two basic elements [5]: mappers and reducers. The idea of this programming model is to design mappers (or map function), which can be used to generate a set of intermediate key/value pairs [4]. The reducer (or reduce function) is used as a shuffling or combining function to merge all of the intermediate values that are associated with the same intermediate key. The main aspect of this algorithm is that if every map and reduce is independent of all other ongoing maps and reduces, and then the operations can be run in parallel on different keys and lists of data.

The process of this approach can be decomposed as follows: (1) Prepare input data: It utilizes the Google File System (GFS or HDFS) as an underlying storage layer to read input and store output [9]. GFS is a chunk-based distributed file system that supports fault tolerance by data partitioning and replication. The big data is divided into small chunks on different worker nodes. (2) The map step: The map function of each node is applied to local data and the output is written to a temporary storage space. (3) The sort and send step: The output from Step 2 is sorted with key such that all data belonging to one key are located on the same node. The sorted results are sent to reduce processors. (4) The reduce step: Each group of output data (per key) is processed in parallel on each reduce node. The user-provided reduce function is executed once for each key value produced by the map step. (5) Produce the final output: This [10] system collects all of the reduce outputs and sorts them by key to produce the final output as shown in Figure 1.

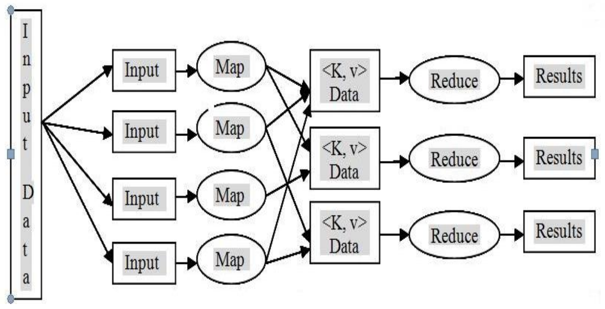

Figure 1. MapReduce Framework

In this paper, the introduction of the most popular distributed clustering algorithms with MapReduce is covered. The goal here is to make a broad and general synthesis concerning the big data clustering with MapReduce issues and pinpoint the advantages of the important techniques. The paper is organized as follows: The second section provides a detailed view of the various distributed clustering techniques dealing with big data's challenges. The last section presents the conclusion.

II. DistributedClustering Algorithms With Mapreduce

Most of the distributed clustering algorithms follow the framework depicted in Figure 2.

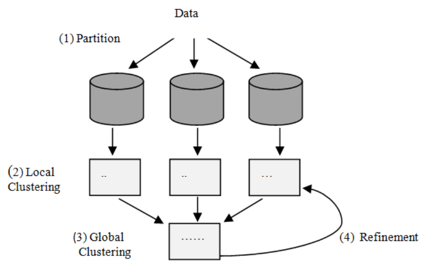

Figure 2. Distributed Clustering Framework

The issue in distributed clustering [1] is the lower accuracy compared to the serial counterpart. There are two main reasons for the lower accuracy. First, each machine performing the local clustering might use a different clustering algorithm than the serial counterpart due to heavy communication costs. Second, even though the same algorithm is used for the local clustering step as well as the serial algorithm, the divided data might change the final aggregated clusters. Here the description of some of the distributed clustering algorithms with MapReduce as follows.

A. K-Means with MapReduce

PKMeans uses MapReduce to distribute the computation of *k*-means[9]. The partitioning is implicitly performed when the data is uploaded to the distributed file system (e.g., GFS or HDFS) of MapReduce. The local clustering is performed in the mapper. The global clustering is performed in the reducer. The computation is demonstrated in Figure 3.

K-Means: Map

```
Input: Data points P, number of clusters c and centroids.
1: for each data point p € P do
2: Assign p to the closet centroid
Output: centroids with associated data points
```

K-Means: Reduce

```
Input: Centroids with associated data points
1: Compute the new centroids by calculating the average of data points in cluster
2: Write the global centroids to the disk
Output: New centroids
```

Figure 3. MapReduce based K-Means

B. Co-Clustering with MapReduce

DisCo is a distributed co-clustering [10] algorithm with MapReduce. The idea is to initiate two Map-Reduce jobs for row and column iterations, and a synchronization step in between to update the global parameters G, r, and c. The map, reduce jobs are illustrated in Figure 4.

CCROWMAPPER (k, v)

```
Globals: Cluster statistics G, labels c
  Source node is $i \equiv k$
  Adjacency list of $i$ is $a_{i:} \equiv V$
  Compute row statistics $g_i := \textsc{RowStatistics}(a_{i:}, c)$
  for each group label $p = 1..k$ do
    if assigning $i$ to $p$ would lower cost then
      $r(i) \leftarrow p$
  emit $\langle r(i), (g_i, \{i\}) \rangle$
```

```
CCROWREDUCER (k, V)
 Row group label is $p \equiv k$
 Initialize $g_p \leftarrow 0$, $I_p \leftarrow \emptyset$
 **for each** map value $(g, I) \in V$ **do**
     $g_p \leftarrow$ COMBINESTATISTICS$(g_p, g)$
     $I_p \leftarrow I_p \cup I$
 **emit** $\langle p, (g_p, I_p)\rangle$

COLLECTRESULTS
 Initialize $G \leftarrow 0$, $r \leftarrow 0$
 **for reduce output** $\langle p, (g_p, I_p)\rangle$ **do**
     $g_{p:} \leftarrow g_p$
     $r(i) \leftarrow p$, for all $i \in I_p$
 **return** G and r
```

Figure 4. MapReduce based Co-Clustering

C. Subspace Clustering with MapReduce

BoW [11] is a distributed subspace clustering algorithm on MapReduce. BoW provides two
Subspace clustering algorithms: Parallel Clustering [7] (ParC) and Sample-and-Ignore (SnI). ParC has three steps as illustrated in Figure 5.
(1) Partition the input data using mappers so that data with the same partition are aggregated to a reducer.
(2) Reducers find clusters in their assigned partitions using any subspace clustering algorithm.
(3) Merge the clusters from the previous step to get final clusters.
Steps (1) and (2) are performed in a map and a reduce stage, respectively, and step (3) is done serially in a machine.

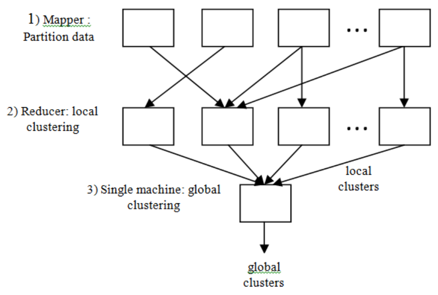
1) Mapper : Partition data
2) Reducer: local clustering
3) Single machine: global clustering
local clusters
global clusters

Figure 5. The ParC algorithm for subspace Clustering

SnI uses sampling to minimize network traffic. SnI comprises two phases [11] as depicted in Figure 6.
In the first phase,
(1) Mappers randomly sample data and send results to a reducer.
(2) The reducer runs a subspace clustering algorithm to find initial clusters.
In the second phase,
(3)Mappers send to reducers only the data points which do not belong to the initial clusters found in the first phase.
(4)The reducers find clusters.
(5)A single machine combines the clusters from the reducer with the initial clusters from the first phase to make final clusters.

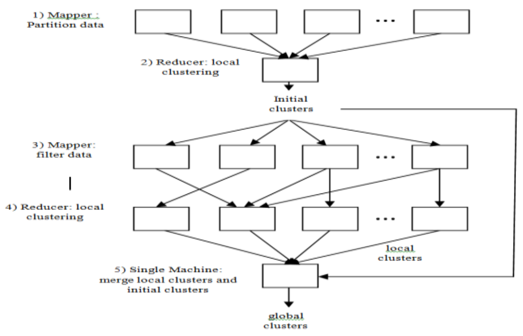
1) Mapper : Partition data
2) Reducer: local clustering
Initial clusters
3) Mapper: filter data
4) Reducer: local clustering
local clusters
5) Single Machine: merge local clusters and initial clusters
global clusters

Figure 6. The SnI algorithm for subspace Clustering

In terms of the number of disk I/Os, ParC is better since it requires only one map-and-reduce step while SnI requires two map-and-reduce steps. However, in terms of the network traffic SnI is better due to the sampling in the first phase and the filtering in the second phase.

III. Conclusion

Traditional centralized clustering algorithms cluster objects stored in a single site, but it cannot satisfy the clustering requirements when objects are distributed. Distributed clustering algorithms can satisfy this need, which extracts a classification mode from distributed objects. MapReduce has gained significant momentum from industry and academia by virtue of its simplicity, scalability, and fault tolerance. In distributed scenario, we can get better performance in terms of memory utilization and speedup if there is utilization of proper blend of resources with MapReduce. This paper analyzes typical distributed clustering algorithms with MapReduce.

References

[1] Xu, Z., & Shi, Y. 2015. Exploring big data analysis: Fundamental scientific problems. Annals of Data Science, 2(4), 363–372.

[2] Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A. Y., & Bouras, A. 2014. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. IEEE Transactions on Emerging Topics in Computing, 2(3), 267–279.

[3] Shim, K. 2012. MapReduce algorithms for big data analysis. Proceedings of the VLDB Endowment, 5(12), 2016–2017.

[4] He, Y., Tan, H., Luo, W., Feng, S., & Fan, J. 2014. MR-DBSCAN: A scalable MapReduce-based DBSCAN algorithm for heavily skewed data. Frontiers of Computer Science, 8(1), 83–99.

[5] Lee K-H, Lee Y-J, Choi H, Chung YD, Moon B (2012) Parallel data processing with MapReduce: a survey. ACM SIGMOD Record 40(4):11–20.

[6] Palit I, Reddy CK (2012) Scalable and parallel boosting with MapReduce. IEEE Trans Knowl Data Eng 24(10):1904–1916.

[7] Zhang Y, Gao Q, Gao L, Wang C (2012) Imapreduce: a distributed computing framework for iterative computation. J Grid Comput 10(1):47–68.

[8] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactons on Pattern Analysis and Machine Intelligence*, 33(3):568–586, 2011.

[9] Weizhong Zhao, Huifang Ma, and Qing He. Parallel *k*-means clustering based on MapReduce. In *CloudCom*, pages 674–679, 2009.

[10] Spiros Papadimitriou and Jimeng Sun. DisCo: Distributed co-clustering with Map-Reduce: A case study towards petabyte-scale end-to-end mining. In *ICDM*, pages 512–521, 2008.

[11] Robson Leonardo Ferreira Cordeiro, Caetano Traina Jr.,

[12] Agma Juci Machado Traina, Julio L´opez, U. Kang, and Christos Faloutsos. Clustering very large multi- dimensional datasets with MapReduce. In *KDD*, pages 690–698, 2011.