Parallelization of DBSCAN algorithm on Spark

Xinyue,ZHANG
HKUST
Big Data Technology
20750194
xzhangfa@connect.ust.hk

Jianming,WANG
HKUST
Big Data Technology
20711227
jwanggb@connect.ust.hk

Chennan,FU
HKUST
Big Data Technology
20710780
cfuac@connect.ust.hk

Zhenhao,XING
HKUST
Big Data Technology
20721648
zxingab@connect.ust.hk

***Abstract*—DBSCAN is an unsupervised learning algorithm used for detecting patterns of clusters based on spatial location and its corresponding density. By identifying these patterns, data can be divided into different clusters which are very useful for people to do some data analysis like finding common features for the corresponding cluster. DBSCAN is widely used in many areas: information retrieval and biological substances creation. However, due to the huge cost of the computation and storage of identifying density for a spatial location, the traditional DBSCAN is hard to be applied in a huge dataset. In order to improve the efficiency of DBSCAN, we try to implement a distributed parallel version of DBSCAN on Spark.**

***Index Terms*—Spark; Data Mining; Parallel Algorithms; DBSCAN Algorithm; Data Partition.**

I. Introduction

With the big data era coming, there is huge data for people to do data mining, which means that people can use the knowledge gained by mining data to improve the efficiency of solving problems or making decisions. Unsupervised learning is a kind of machine learning algorithm to do data mining without labels, making the machine learn the relationship by identifying the potential relationship from features of data points. In unsupervised learning, clustering algorithms are used for helping people to category data. There are four main types of clustering: partitioning based, hierarchy-based, grid-based, and density-based. In this project, we focus on improving the density-based clustering algorithm: DBSCAN (Density-based spatial clustering of applications with noise) algorithm.

A. Sequential DBSCAN

1) Concept: E neighbourhood, the area within the radius of the given object is called E neighbourhood. Core point, the core given radius possesses sample points which are more than the given minimal points is called the core point. Non-core point(border point): the points in the cluster which are not the core points. Cluster, If p is a core point, it forms a cluster with all the points reachable by it including core points and non-core points. Minimal points, the number is controlled by users to define the minimal number of one cluster except for the core point. Reachability, if the sample point q in the E neighbourhood of core point p, then point q is directly reachable to point p. Outliers(noise point), all the points that can't be reachable by any other point are called outliers.

2) Description: DBSCAN requires two parameters - ϵ and the minimum number of points (MinPts) required to form a high-density area. It starts from an arbitrary unvisited point and then explores the E neighbourhood of this point, If there are enough points which are more than the minPts in E neighbourhood, create a new cluster, otherwise, this point will be labelled as noise. Note that the point may be found in the E neighbourhood of other points.

3) Pipeline: Input - sample points D, MinPts, ϵ. Output - cluster set. All points in the D are marked unprocessed. For each point P in the D, check the E neighbourhood and whether there are more than MinPts points in it. If the points number in the E neighbourhood less than MinPts, mark the P as an outlier. Otherwise mark P as a core point, create a new cluster C and all the points in $P's$ E neighbourhood will be added in C. Repeat until all the points have been processed.

II. Parallelized DBSCAN

Existing research on the DBCAN algorithm - the whole process of DBCAN on Spark is divided into two parts - partitioning and fusion.

A. Existing Model

1) Partitioning: Basically, it continuously groups into the same cluster the points that are near each other (or within a certain distance from each other). The following is the process of how to do this on a distributed, share-nothing platform. Given the original points, a virtual fishnet is cast on the point space. The points in the same cells are processed together in a distributed matter. This togetherness is achieved by mapping the point coordinates (x,y) to the cell lower-left corner (r,c). All the points that have the same (r,c) are grouped together and locally processed on different nodes. When the point space is partitioned based on cells and executed on different nodes, the edge points do not "see" all their neighbours as they are not in the processing partition.

2) Fusion: All points in a cell that are away from the edge are emitted to the neighbouring cells, in such that now, we can perform per node a local DBSCAN. This double processing of the same edge point by two nodes is actually a blessing, as we can now relate and merge two neighbouring local clusters into one global one. Note: The neighbourhood search in SpatialIndex is based on a bounding box search where the box is centred about the given point and the width

and height of the box is ϵ. All the points that fall into the box are considered neighbours.

B. Improved Plan

1) Calculate the distance between two sample points: Unlike conventional DBCAN-Spark work considers a virtual fishnet to do partition, this project considers dividing the partitions with different sample points into multiple batches to transmit to the Driver end, and then broadcast to each executor to calculate the distance separately, and union the final result to indirectly realize the double traversal. In order to reduce the amount of calculation, the spatial index Rtree is constructed before broadcasting to accelerate.

2) Merge connected temporary clusters to get clusters: First, merge the temporary clusters within each partition, then reduce the number of partitions to repartition, and then do the first step again. This process is repeated continuously to realize that all temporary clusters are included in one partition, and the merging of all temporary clusters is completed.

III. Plan on project

Finishing some research on related papers, a step by step plan for the DBSCAN algorithm on Spark is constructed. Implementing the original DBSCAN algorithm in python is needed, which can help gain an overall understanding of the basic idea, and also, it will be used for the comparison with the DBSCAN algorithm on Spark later.

A pipeline is created for our project and several major parts will be involved:

- In a distributed environment, data points are distributed into different RDDs, thus, it is hard to loop all the data and compute the distance among them. Hence, a new way of computing the distance between two data points will be applied.
- As for conducting temporary clusters, counting the number of data points within the radius R of the neighbouring neighbourhood. And record their IDs, if the number of these data points exceeds min-points, construct temporary clusters and maintain a list of core points.
- The main problem is (in a distributed environment) how to link the temporary clusters together as they are stored in different RDDs. Firstly, merge the temporary clusters within each RDD partition, then reduce the number of partitions to repartition, after that, merge each temporary cluster within each partition again. This process is repeated continuously, and finally, all the temporary clusters are formed into one RDD partition, all of them are merged.

Meanwhile, the sample datasets used are synthetic data, which are generated according to some distribution. Also, we will grab some popular datasets from Kaggle or UCI to test the performance of the algorithm. Furthermore, as the whole project was implemented in Azure, HDFS environment will be conducted and all the sample datasets will be stored in it, by doing this, each time reading data via HDFS will be parallelized, which can be more efficient.

Algorithm 1: Parallelized DBSCAN on Spark

Input: sample points D, MinPts, ϵ
Result: cluster set
1. Read an input file from HDFS and generate RDDs from the read data.
2. Transform the existing RDDs into appropritate RDDs with Point type.
3. Distribute those RDDs into excutors;
for *closure'start* **do**
  Mark point p as vistied
  Let N be E-neighborhood of p
  Pull all the data points to Driver in several batches and then broadcast to each Excutor to calculate the distance, and then union the result
  Using Rtree to accelerate the distance computing;
  if *the size of N<minpts* **then**
    Mark p as noise;
  else
    Create a cluster C, and add p to C;
    for *point p in N* **do**
      if *p is unvisited* **then**
        Let N be the E-neighborhood of p;
        if *size of N is $\geq minpts$* **then**
          Add those points to N;
        end
      end
      if *p is not yet member of any* **then**
        Add p to C;
      end
    end
  end
end
while *number of closure is larger than one* **do**
  for *closure* **do**
    merge temporary clusters;
  end
  Reduce number of closures to partition again;
end

TABLE I
Pseudo Code of DBSCAN on Spark

References

[1] Ester, M., Kriegel, H. P., Sander, J., Xu, X. (1996, August), "A density-based algorithm for discovering clusters in large spatial databases with noise", In Kdd (Vol. 96, No. 34, pp. 226-231).

[2] Birant, D., Kut, A. (2007). ST-DBSCAN: An algorithm for clustering spatial–temporal data. Data knowledge engineering, 60(1), 208-221.

[3] Ester M, Kriegel H P, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise[C]//Kdd. 1996, 96(34): 226-231.

[4] Patwary M, Palsetia D, Agrawal A, et al. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure[C]//High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for. IEEE, 2012: 1-11.