Parallel Implementation of DBSCAN Algorithm Based on Spark

PENG Qiulei **WANG Jue** **LIU Jianyi**

Abstract

In this project, the parallel implementation of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) was realized with improved splitting algorithm based on Spark in HDFS clusters, which solved the operation problem of big dataset. The performance of parallel DBSCAN algorithm and the sequential DBSCAN algorithm were compared, and the influence of different parameters on clustering results was analyzed.

1 Introduction

In the case of unlabelled data, it is a hot topic to discover the underlying information in the unlabelled data, which is unsupervised learning. Clustering is a kind of unsupervised learning by grouping similar objects into the same cluster, so that the similarity of data objects in the same cluster and the difference of data objects not in the same cluster are as large as possible. In the real life, many situations are suitable to use this kind of algorithm. For instance, it is useful to classify the customers according to their records of consumption and give them different recommendations in the future.

DBSCAN is a kind of representative density-based Clustering algorithm, and this kind of density clustering algorithm generally assumes that the category can be determined by the density of the sample distribution. Even though the dataset contains noise, this algorithm can still discover clusters of different shapes and sizes from a large amount of data.

If the sample set is large, the clustering convergence time of DBSCAN is too long. In our project, SPARK is used to alleviate this problem and make the program more efficient. Based on the improved splitting algorithm, the parallel algorithm is realized in SPARK. Finally, the performance of sequential DBSCAN and SPARK DBSCAN in HDFS cluster are compared, and the later has better effect.

2 Description of Serial DBSCAN

2.1 Basic rules of DBSCAN

DBSCAN is Density-Based Spatial Clustering of Applications with Noise. In this algorithm, the points would be classified into 3 types as Figure 1 shown below, the core points, border points, and outliers, according to the definitions of concepts, directly-density-reachable (DDR), density-reachable(DR), and density-connected(DC). And it is based on two important parameters of DBSCAN, one is MinPts, the Minimum number of points in an Eps-neighborhood of that point, the other is Eps, the Maximum radius of the neighborhood.

If point q is within the ε-Neighborhood of p and p is a core point, the point q is directly density reachable from object p. If there are a set of core points leading from p to q, the point q is density reachable from p. If both p and q are density-reachable from another point m, these two points are density-connected. Directly density reachability and density reachability are not symmetric, while density connectivity satisfies the symmetry.

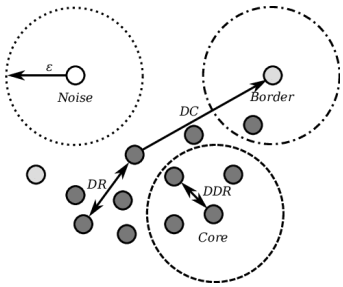

Figure 1: DBSCAN Demo

Figure 2 is a mind map about some concepts for DBSCAN, the clustering procedure is mainly based on these concepts and parameters. Different parameter combinations sometimes have very different results, and the clustering results are sensitive to these parameters. The greatest strength of DBSCAN is that the number of clusters does not need to be specified.

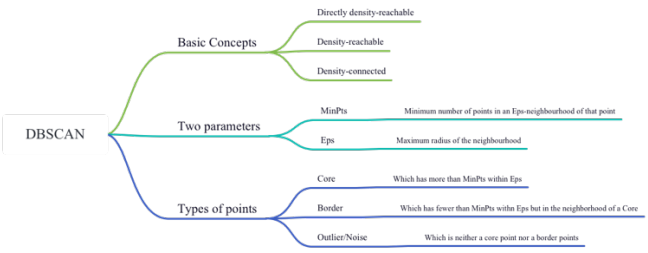

Figure 2: DBSCAN Mind Map

2.2 Procedures of DBSCAN

Figure 3 is the flow chart about DBSCAN. It obviously can be seen that this algorithm collects points as clusters (points with many close neighbors) and marked stand-alone anomalous points(Whose nearest neighbor is too far away) as outliers in a low-density region.

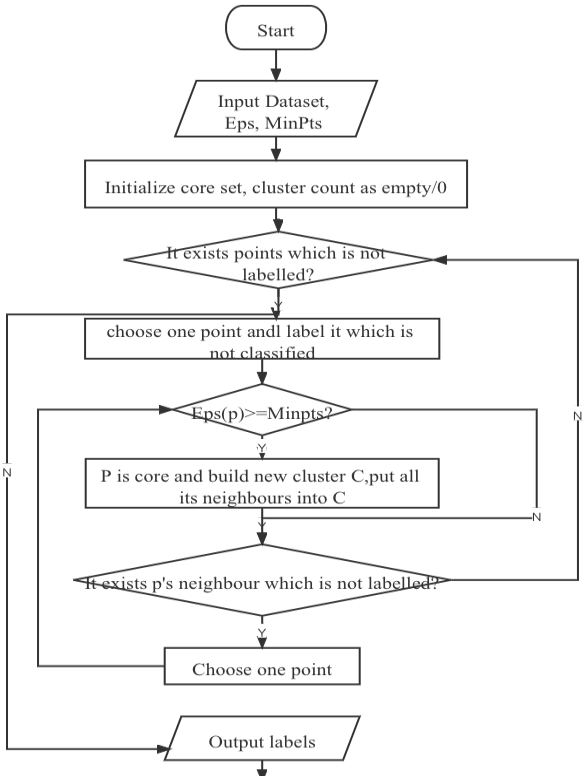

Figure 3: DBSCAN Flow chart

According to the feature of DBSCAN, it will compute the distance of every two points, when there is a large amount of data, the processing speed is very slow, and the consumption is very high. But usually, adjusting the parameters is a repeatedly work to find the best parameters, so it is time-consuming for big dataset.

This is the pseudocode shown in Algorithm 1.
D:$(x_1, x_2, \ldots, x_m)$ a dataset including m datapoints.
ϵ: a distance threshold used when calculating the neighbours of a data point.
Minpts: the minimum number of points to be clustered.
distance Measurement: way to calculate the distance of two points. In this project, this is defaulted as Euclidean distance.

Algorithm 1: DBScan Algorithm

Input : D, ϵ, *Minpts*
Output: a set of clusters

```
coreSet ← empty set // let core object set be a empty set
k ← 0 // initial number of clusters
unvisited ← D
clusters ← empty set
for j ← 0 to m − 1 do
    D[j]_neighbours ← set of neighbours within threshold ε
    if len(D[j]_neighbours) >= Minpts then
        coreSet ← coreSet ∪{D[j]}
    end if
end for
while coreSet is not empty do
    o ← random select from coreSet
    queue_cur ← {o}
    k ← k + 1
    Ck ← {o}
    unvisited ← unvisited−{o}
    while True do
        if queueCur is empty then
            add Ck to clusters
            coreSet ← coreSet− Ck
            break
        else
            coreSet ← coreSet− Ck
            o' ← select from queue_cur
            o'Neighbours ← set of its neighbours within threshold ε
            delta = o'Neighbours ∩ unvisited
            Ck ← Ck ∪ delta
            unvisited ← unvisited − delta
            queueCur ← queueCur ∪ (delta ∩ coreSet) - o'
        end if
    end while
end while
return clusters
```

Algorithm1: DBSCAN Algorithm

3 Implementation of Parallelization

To implement the parallelization into DBSCAN algorithm, this project introduced divide-and-conquer in practice. The main procedure to integrate spark and DBSCAN algorithm can be divided into three steps, separating dataset into several partitions, executing DBSCAN for each partition individually, and merging the result of each partition together.

Considering the core of DBSCAN is finding core samples of high density and expands clusters from them, the location of each data point is significant in the clustering process. If datasets were separated randomly, then merging the results from each partition still need DBSCAN to cluster those results. However, keeping the data points with their neighbor together can improve the merging part by only analyze the datapoints on the boundary of each partition space. In addition, since DBSCAN complete clustering tasks by computing the distance between points, using the location of data points to separate datasets can group most adjacent points together, and make the local DBSCAN more efficient. Thus, partitioning data by coordinates is the strategy implied in this project.

3.1 Evenly Splitting

Evenly Splitting is a method that splitting the data space into four section based on the (x,y) coordinates. Please see the Figure 4 for the reference. Datasets were analyzed and separated into 4 parts according to the coordinates of features. Using midrange of each feature as the boundary to split the data into four parts. Nevertheless, a problem occurs when for merging. The ground truth cluster crossing over the partition might be mis-classified into multiple clusters. To solve this problem, partitioning was changed into another way showing in the Figure 5. Partitions shared overlap region to own common data points, the range of overlap region is $\pm\epsilon$. Under this situation, for all points in multiple partitions at the same time were labeled into multiple clusters (0 ,1, 2, ⋯), that means points in those clusters should be grouped together into one cluster. After processing all points labeled with multiple clusters, the merging step is complete.

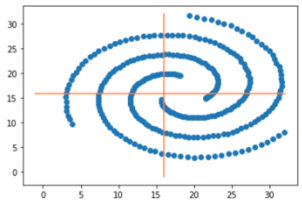

Figure 4

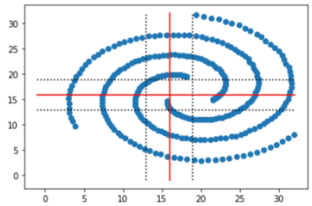

Figure 5

3.2 Improved Splitting

Improved Splitting by space and number of points in space is a method to partition data based on both location and the number of points in the space. Evenly splitting had some limitation during the experiments. For example, when data was not normally distributed in the whole data space, the workload distributed to each partition was unbalanced, which produced inefficiency. Therefore, new method was implied into the experiment. The main idea of this strategy was splitting by space with an addition condition that each space divided should have almost same number of data points. Figure 6 shows the detail to partition dataset.

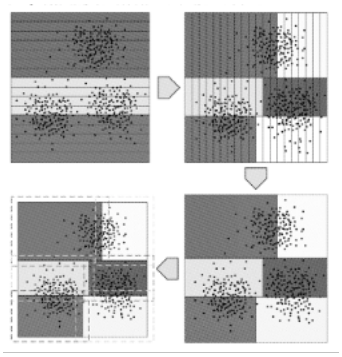

Figure 6

There was a new hyperparameter, number of spaces to be partitioned in a dimension, to be set. It divided the one dimension into multiple section $\{s_1, s_2, s_3, \dots\}$ with width of 2ϵ. Then starting from s_1, the number of datapoint in the section was accumulated until satisfied the threshold to build a subspace. This step was repeated to find all subspaces. And the whole procedure was implied for each subspace found to find sub-subspace in another dimension. The pseudo code is also provided.

Algorithm 1: Partitioning Algorithm

Input : D, ϵ, *numSpaces*
Output: intervals of data spaces

$M \leftarrow$ size(D) $N \leftarrow$ M /numSpaces intervals $\leftarrow$ empty list
pmin $\leftarrow$ minimum value in current dimension
pmax $\leftarrow$ maximum value in current dimension
$s \leftarrow$ (pmax-pmin)/(2*ϵ)
start $\leftarrow$ pmin
end $\leftarrow$ start+2*ϵ
while *start* $<$ *pmax and end* $<$ *pmax* **do**
  count $\leftarrow$ number pf data points in the the region [start, end]
  if *count* >= *size*/*numSpaces* **then**
    add (*start*, *end*) to intervals
    start $\leftarrow$ *end*
    end $\leftarrow$ *start* + 2 * *eps*
  else
    end $\leftarrow$ *start* + 2 * *eps*
  end if
end while
add (*start*, *end*) to intervals
return *intervals*

Algorithm2: Partitioning Algorithm

4 Experiment

In this project, HDFS is used as A storage system platform to test the performance of the parallel algorithm in Spark cluster environment, and the comparison experiment is made with the single machine ALGORITHM. The HDFS Cluster is built in Microsoft Azure, and the flowchart parallelization is shown below.

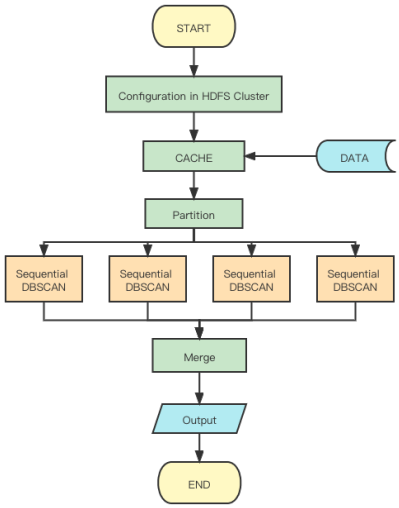

Figure 7 Parallelization Flowchart

4.1 Experimental environment

The experimental cluster includes 1 MASTER node and 3 Slave nodes. The configuration of each calculation node is the same. The cluster environment is listed in Table 1. And different datasets in Table 2 are used to test the Algorithm clustering effect, and different parameter combinations are used. In datasets with different types, parallel DBSCAN and Sequential DBSCAN are compared according to the running time with the same parameters.

Configuration	Parameter
Operating system	Linux
Size	Standard D2s v3 (2 vcpus, 8 GiB memory)
JDK Version	1.8.0_292
Hadoop Version	2.7
Spark Version	3.0.1
Python Version	3.8.10

Table1 Configuration Parameter

Dataset Name	Dataset Size	Cluster number	Distribution characteristics
Spiral	312 rows* 2 Dimension	3	Spiral shape, uneven density
R15	600 rows * 2 Dimension	15	Globose, uneven density
Test_1k	1000 rows * 2 Dimension	2	Globose, uniform density
Test_1500	1500 rows * 2 Dimension	3	Globose, uniform density
Test_2k	2000 rows * 2 Dimension	3	Globose, uniform density

Table 2 Dataset Description

4.2 The influence of parameters on clustering.

The minimum neighborhood radius and the minimum number of points contained in the neighborhood are the two most important parameters in SparkDBSCAN algorithm, which are crucial to the clustering accuracy and the number of clustering, so observing the clustering effect under different parameters is very necessary for the clustering effect of SparkDBSCAN algorithm.

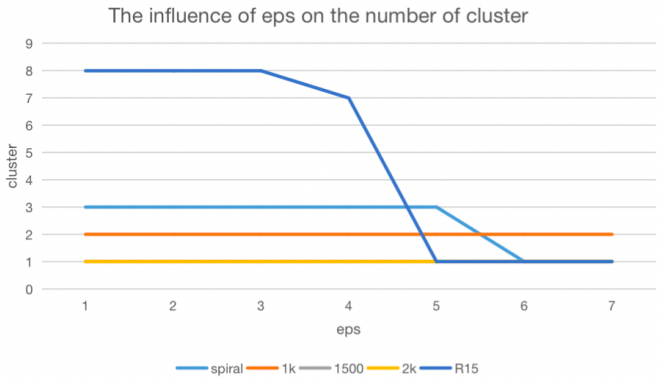

Figure 8

It can be seen from Figure 8, the change of the number of clustering shows a certain degree of heterogeneity. The number of R15 and spiral data sets decreases after holding a certain number of clusters. Spiral data sets with fewer clusters show a decreasing trend, while the number of R15 clustering keeps decreasing until it is stable. For a certain data set, even if the number of clustering generated when it is consistent with the number of clustering itself (such as 2k and 1k), the clustering accuracy may not be very high, because a small Eps will produce more noise points, thus reducing the accuracy.

From the above analysis, it can be concluded that generally Eps has a role in controlling the number of clustering. Selecting suitable Eps can avoid generating too many noise points and misclassification points.

4.3 Parallelization Visualization

The Figures below show the clustering results in different partitions after the splitting operation in spiral dataset.

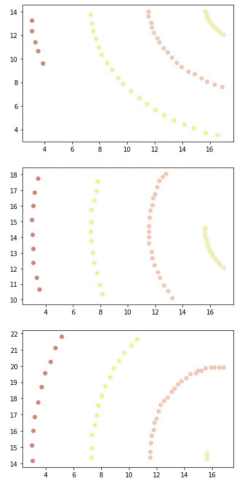

Figure 9

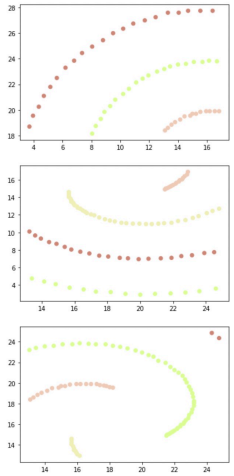

Figure 10

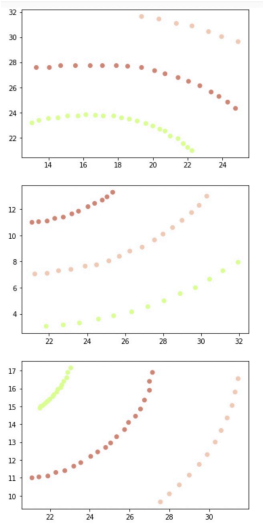

Figure 11

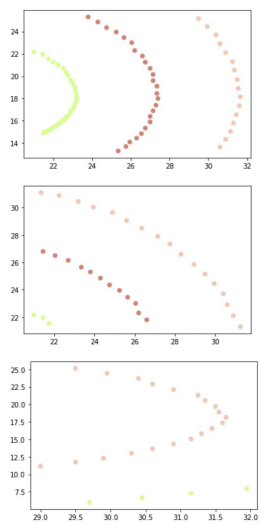

Figure 12

The result after merging all partitions is shown below.

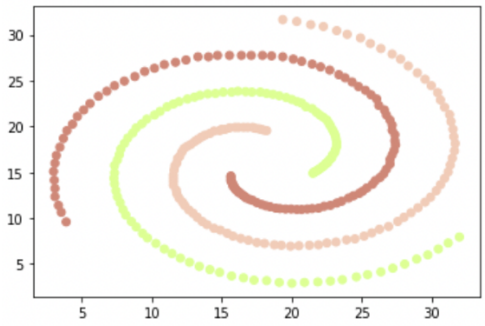

Figure 13

4.4 Performance Comparison

In this part, the performance of Sequential DBSCAN and SparkDBSCAN in different datasets are compared, according to the running time in different dataset sizes.

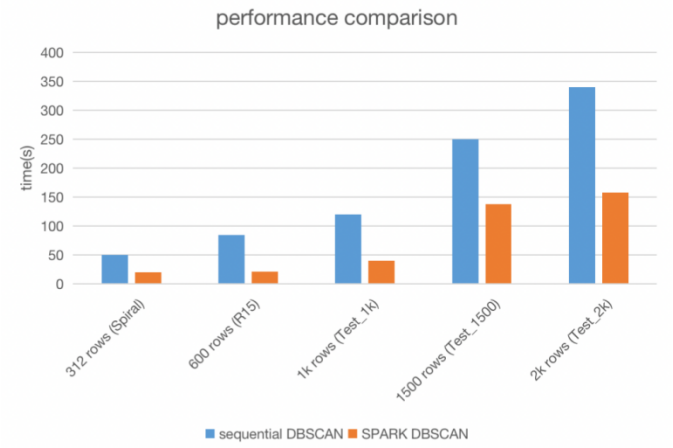

Figure 14 run time comparison in different dataset sizes

It can be seen that the running time of the Sequential DBSCAN algorithm exhibits a quadratic curve growth as the amount of data increases. This is mainly because the time complexity of the DBSCAN algorithm is $O(n^2)$, and the SparkDBSCAN algorithm is due to its parallelism, each computing node only shares part of the data. Relatively speaking, it is not very sensitive to the increase in the amount of data, so its growth rate is not obvious. When the amount of data increases to the larger order of magnitude, the advantages of the Spark DBSCAN algorithm is more obvious and Sequential DBSCAN could even be executed successfully in one single machine.

As the amount of data size grows, it is possible that the training time of larger dataset is lessthan that of smaller dataset as shown in Figure 14 above. The reason is that the density of data points is larger in bigger dataset, and it might be easier in clustering to some extent.

5 Conclusion

This project implemented DBSCAN algorithm in parallel based on Spark framework and used improved splitting algorithm to shorten the execution time of the algorithm. The clustering quality and execution time in SparkDBSCAN and sequential DBSCAN algorithm are compared experimentally, and the effects of different parameters on clustering results are analyzed. The experimental results show that Spark parallelized DBSCAN algorithm achieves ideal running time. The power of parallel programming implemented by Spark is demonstrated in this experiment.

6 Contribution

In this section, we would like to share the contribution distribution in our project. PENG, Qiulei implemented local DBSCAN coding part, and WANG, Jue implemented the integration of pySpark and local DBSCAN. LIU, Jianyi conducted experiments comparison. As to the project report, WANG and PENG finished the introduction, algorithm analysis, experiments analysis and conclusion, LIU filled in experiments data and made excel plots.

7 References

[1]. HUANG, Ming-ji, and Qian ZHANG. Design and Implementation of Parallel DBSCAN Algorithm Based on Spark. Nov. 2017, www.jsjkx.com/CN/article/openArticlePDF.jsp?id=16499.

[2]. WANG,SHEN and Jinyu.Liu,Parallel Implemetation of DBSCAN Algorithm Based on Spark. cse.hkust.edu.hk/msbd5003/pastproj/deep1.pd

[3]. Luo G, Luo X, Gooch T F, et al. A parallel dbscan algorithm based on spark[C]//2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking

(SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom). IEEE, 2016: 548-553.
[4]. Arlia D, Coppola M. Experiments in parallel clustering with DBSCAN[C]//European Conference on Parallel Processing. Springer, Berlin, Heidelberg, 2001: 326-331.