2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)

# A Parallel DBSCAN Algorithm Based On Spark

Guangchun Luo[1]; Xiaoyu Luo[1]; Thomas Fairley Gooch[2]; Ling Tian[1]; Ke Qin[1]
[1]School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
[2]Department of Computer Science, Georgia State University, Atlanta, USA
gcluo@uestc.edu.cn; 398814969@qq.com; thomasfgooch@gmail.com; ruan052@126.com;qinke@uestc.edu.cn

***Abstract*— With the explosive growth of data, we have entered the era of big data. In order to sift through masses of information, many data mining algorithms using parallelization are being implemented. Cluster analysis occupies a pivotal position in data mining, and the DBSCAN algorithm is one of the most widely used algorithms for clustering. However, when the existing parallel DBSCAN algorithms create data partitions, the original database is usually divided into several disjoint partitions; with the increase in data dimension, the splitting and consolidation of high-dimensional space will consume a lot of time. To solve the problem, this paper proposes a parallel DBSCAN algorithm (S_DBSCAN) based on Spark, which can quickly realize the partition of the original data and the combination of the clustering results. It is divided into the following steps: 1) partitioning the raw data based on a random sample, 2) computing local DBSCAN algorithms in parallel, 3) merging the data partitions based on the centroid. Compared with the traditional DBSCAN algorithm, the experimental result shows the proposed S_DBSCAN algorithm provides better operating efficiency and scalability.**

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

## I. INTRODUCTION

With the current innovative information technology, data is showing explosive growth, many massive data-based applications have been quite popular. Internationally, Facebook users number about one third of the global population. In China, 2015 Tmall double eleven carnival activities achieved the turnover of nearly 100 billion yuan, and created a user's transaction volume of up to 85,900 strokes/sec. It seems that the internet industry is facing a huge shift from IT to DT (Data Technology). How to improve the ability to mine knowledge from massive amounts of data has become a problem. In order to find out the differences and connections between data, data mining[1] appears in the horizon as a new discipline; and is applicable in varied industries.

Cluster analysis occupies a pivotal position in data mining, and has received wide attention. Clustering is generally in accordance with a certain similarity measure, so that highly similar data can be gathered together. From the point view of machine learning, clustering belongs to the unsupervised learning process, and automatically finds relations between data. Cluster analysis plays an important role in many fields including statistics, image processing, pattern recognition, and psychology.

Among clustering algorithms, Density-based Spatial Clustering of Applications with Noise (DBSCAN)[2] is one of the most widely used. Its popularity stems from the benefits that DBSCAN provides over other clustering algorithms: the ability to discover arbitrarily shaped clusters, robustness in the presence of noise and simplicity in its input parameters (unlike other clustering algorithms, DBSCAN does not require the user to enter the number of clusters to be found prior to execution). DBSCAN is also insensitive to the order of the points in the dataset. As a result, DBSCAN has achieved great success and has become the most cited clustering method in relevant scientific literatures.

In order to deal with massive data, the literatures have proposed multiple approaches for the parallel execution of DBSCAN. Xiaowei.Xu proposed PDBSCAN by distributed R* tree to improve the efficiency of the algorithm[3].Ali Patwary proposed an algorithm named PDSDBSCAN using Disjoint-Set-based data structure, it has broken the ordering of data access, resulting in a better load balancing[4].Benjamin Welton proposed a GPU based parallelization DBSCAN with significantly improving efficiency through effective data partitioning[5]. MapReduce was introduced in 2004 in a seminal paper published by J. Dean and S. Ghemawat[6]. B. R. Dai, and I. C. Lin in [7] and Y. He, et al. in [8] both applied their parallel DBSCAN algorithm based on MapReduce.

While the existing parallel DBSCAN algorithms can overcome some of the problems by traditional DBSCAN algorithms, it also has the following disadvantages:

1)When the existing parallel DBSCAN algorithms make data partitions, the original database is usually divided into several disjoint partitions, with the increase in data dimension, the splitting of high-dimensional space will consume a lot of time.

2)When doing partition merging, we need to do boundary determination for 2m times for each partition, where m is the dimension of data, which will definitely consume a lot of time and result in low efficiency.

In order to solve the problems of existing parallel DBSCAN algorithms and their low efficiency when doing data partitions and partition merges, this paper proposes a parallel DBSCAN algorithm(S_DBSCAN) based on Spark[9][10]; which could quickly realize these merges and divisions of clustering results of the original data.

The work of this paper is as follows:

- Describing the S_DBSCAN algorithm that takes full advantage of Apache Spark's parallel capabilities and

This work is supported by the Science and Technology Department of Sichuan Province (No.2014FZ0087).

978-1-5090-3936-4/16 $31.00 © 2016 IEEE
DOI 10.1109/BDCloud-SocialCom-SustainCom.2016.85

obtains the same results as the original DBSCAN algorithm.

- Implementing S_DBSCAN algorithm on Spark, while showing the details of S_DBSCAN.
- Evaluating the performance of S_DBSCAN using outpatient medical data to solve practical problems and to verify the capability of parallelization.

## II. SPARK AND DBSCAN ALGORITHM

### A. Spark Overview

Spark is a new framework of distributed parallel processing launched by the Apache Foundation, which is very suitable for parallel distributed computing with massive data. Spark overcomes some of MapReduce's shortcomings and has all its advantages at the same time. To ensure the reliability of the results, MapReduce needs to write back every intermediate calculation to HDFS [11], which results in low efficiency, while Spark saves intermediate calculations in memory task to improve processing speed. Due to the memory-based features, Spark is suitable to data mining and machine learning algorithms which require multiple iterations. Resillient Distributed Dataset (RDD) [12] is a distributed-stored dataset, and it is an abstract application of distributed memory. Spark is an implementation of the RDD's abstraction, providing a programming framework that allows the programmer to process large amounts of data, and also to take advantage of volatile memory in order to speed-up computation. Spark has two important attributes worth mentioning; a Narrow sense and a Broad sense. Narrow Spark refers to a calculation framework of data processed layers. Broad Spark refers to Spark ecosystems. Spark uses memory-based computing, which is called resilient distributed datasets RDD; as mentioned previously. Spark has also developed a high-throughput memory system called Tachyon; an additional upper layer with similarities to Hadoop in the Mahout machine learning library MLlib, a diagram calculation called GraphX, Shark SQL, and so on. Spark architecture is shown in Figure 1

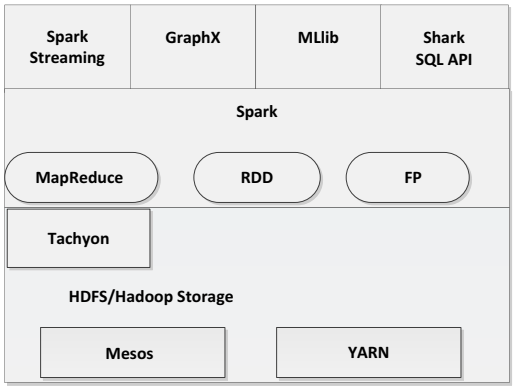


Figure 1. Spark architecture

RDD is an abstract application of distributed memory. If a RDD data fragment is lost, Spark could reconstruct the missing fragment according to a coarse-grained log data called Lineage, which will undoubtedly improve the fault tolerance of RDD. RDD is read-only, and can be serialized by a persist or a cache function of the cache in memory, thus reducing the amount of disk IO, which greatly improves the efficiency of machine learning algorithms. RDD could be created in two ways, one is created from memory by a parallelize method, the other is from a file created by textFile from HDFS, HBase, Amazon S3 and Cassandra. Spark in most computing interfaces are RDD-based, therefore RDD input format determines the output format of the Spark programming interface. Spark in the RDD provides two types of operators: transformation and action. Transformation is to generate a new RDD from an existing RDD. Action is to write results to HDFS through a user-defined function. Each transformation operator does not do real calculations until it meets an action operator. When the transformation conversion path is long, you can use the checkpoint operation to write data to the storage system. The Spark programming model is shown in Figure 2.

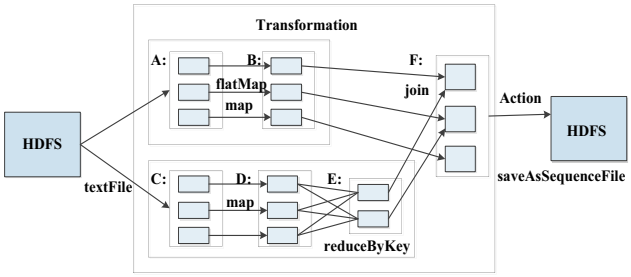


Figure 2. Spark programming model

The Spark system architecture is typical of the Master-Slave structure, consisting of a Master node and several Worker nodes. The user-written Spark program is called Driver, users can achieve RDD conversion operation by Driver and interact with the Master node.

### B. The DBSCAN Algorithm

The purpose of a clustering algorithm is to divide a mass of raw data into separate groups (clusters) which are meaningful, useful, and efficiently accessible. DBSCAN and K-means are two common techniques to deal with clustering problems.

DBSCAN is a density-based clustering algorithm. Density-based clustering algorithms define a cluster as an area that has a higher data density than its surrounding area. In DBSCAN, density is measured by analyzing whether a point has at least a minimum number of points ( $MinPts$ ) inside a given radius ( $\varepsilon$ ). That is, if the $\varepsilon$ neighborhood (points within $\varepsilon$ )of a given point has exceeded a particular density threshold ( $MinPts$ ) at that given point; then the point and its neighbors, form a cluster. There are a few key concepts that stem from the definition of DBSCAN that bear further explanation:

- $N_e(p)$ : for a point $p \in X$ ,its $\varepsilon$ neighborhood is defined as $N_e(p) = \{x \in X \mid dist(x, p) \le \varepsilon\}$ where dist is the distance function.

- core point: a point $p \in X$ is a core point if $N_e(p) \geq MinPts$
- directly density reachable: a point $p \in X$ is directly density reachable from $x \in X$ if $p \in N_e(x)$ and x is a core point.
- density reachable: a point $y \in X$ is density reachable form $x \in X$ if there is a chain of points $p_1, p_2, ...p_n$ where $y = p_1$ and $x = p_n$ and each $p_n \neq y$ is directly density reachable from x.
- cluster: C is a cluster in X if $C \subset X$ and for any $p, q \in C$ ,p and q are density connected.
- border point: p is a border point, if $p \in cluster$ C and p is not a core point.
- noise point: p is a border point if it does not belong to any cluster.

DBSCAN algorithm process is as follows:

1)Determine the radius $\varepsilon$ , minimum number of neighborhood objects $MinPts$.

2)Selecting an arbitrary point p from D to start neighborhood searching, if $N_e(p) \geq MinPts$ then mark p as core point, otherwise mark p as noise point.

3)if p is a core point, then make a cluster c by p, adding all of point belong to cluster c to a list for recursive calls.

4) Repeat 2), 3) until all of the objects in the collection of data objects are marked, and return to the cluster as a class, or to identify those objects do not belong to any kind of noise cluster.

DBSCAN algorithm uses Euler distance to measure distance. For two data $x_i$ and $x_j$ with m dimension, the formula to measure distance is as follows:

$$d(x_i, x_j) = \| x_i - x_j \| = [\sum_{k=1}^{m} | x_{ik} - x_{jk} |^2]^{\frac{1}{2}} \quad (1)$$

## III. THE PROPOSED S_DBSCAN ALGORITHM

In this section, the paper focuses on the solution of finding density-based clusters from raw data on the Spark platform. In order to solve the problems by the existing parallel DBSCAN algorithms; e.g., low efficiency when doing data partition and combination; this paper proposes a parallel DBSCAN algorithm(S_DBSCAN) based on Spark, which could quickly realize mergers and divisions of clustering results from the original data. The S_DBSCAN algorithm is divided into the following steps: partitioning the raw data based on random samples, computing local DBSCAN algorithms in parallel and merging the data partitions based on the centroid. The S_DBSCAN algorithm's framework is shown in Figure 3.

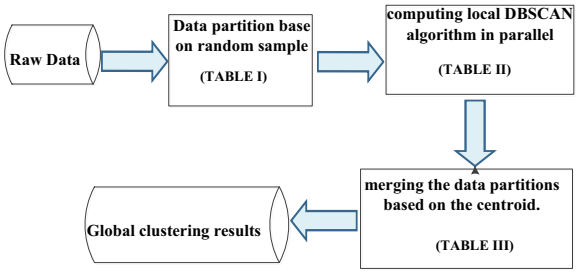


Figure 3. S_DBSCAN algorithm's framework

### A. Data partition based on random sample

To be able to effectively divide the original data, this paper proposes a data partition method based on a random sample. The idea is as follows: Firstly, determine the number of partitions based on the actual computing nodes; and on this basis, through a custom random function, the original random data would export to each slice. Each slice of data points has roughly the same amount of data. Each slice is equivalent to a simple random sampling. When the number of samples extracted by each slice is large enough, it has a similar distribution with the raw data. This paper requires Data_PartitionMap and Data_PartitionReduce to complete the data partition. The procedure of data partition is shown in TABLE I

TABLE I. Procedure of data partition

| |
|---|
| **Step1(Data_PartitionMap):** |
| Input: <ID, Raw_data> |
| Output: <key, Raw_data> |
| Process: |
| 1)Reading every raw data, and get the value. |
| 2)Using random function to generates a random number between 1 to k. |
| 3)Setting the random number number as a key, the output <key, Raw Data>. |
| **Step2(Data_PartitionReduce):** |
| Input: <key,Raw_data> |
| Output: <key, list(Raw_data)> |
| Process: |
| 1)The Raw_data will be distributed to the same Reducer in order to complete the division of the original data. |

### B. Computing local DBSCAN algorithms in parallel

The input of this stage is the output of last stage. The purpose of this stage is to compute a local DBSCAN algorithm in parallel thus creating partial clustering results. This paper requires Local_DBSCAN_Map and Local_DBSCAN_ReduceByKey to generate partial cluster. The Schematic diagram of Computing local DBSCAN algorithms in parallel is shown in Figure 4.

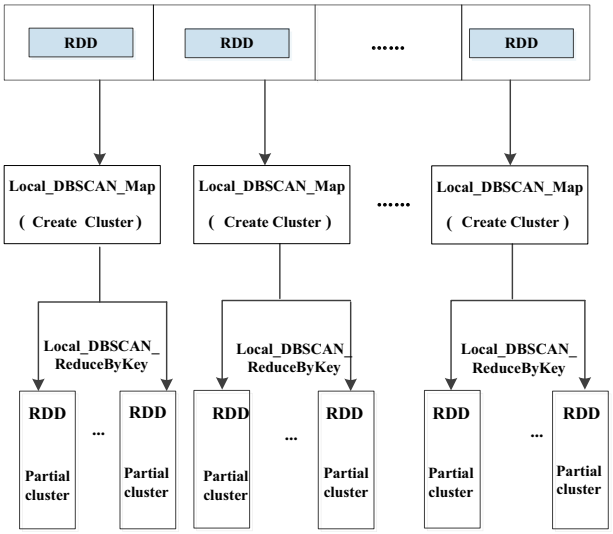


Figure 4. The Schematic diagram of Computing local DBSCAN algorithms in parallel

For the map task, we will finish generating a partial cluster, and for the reducebykey task, a partial cluster will be saved as a new RDD to HDFS and every centroid for the partial cluster will be calculated. The procedure of computing the local DBSCAN algorithm is shown in TABLE II

TABLE II The procedure of computing local DBSCAN algorithm

**Step1(Local_DBSCAN_Map):**

Input: <key, list(Raw_data)>, $\varepsilon$ , *MinPts* / *k*

Output: <CID, (Flag,Raw_data)>or<Flag,Raw_data>

Process:

1)Reading every raw data, and get the value list(Raw_data).

2) Selecting an arbitrary point p from Raw_data to start neighborhood searching, if $N_\varepsilon(p) \geq MinPts$ then mark p as a core point, otherwise mark p as noise point.

3) if p is a core point, then make a cluster c by p, adding all of the points belonging to cluster c to a list for recursive calls.

4) Repeat 2), 3) until all of the objects in the collection of data objects are marked, and return to the cluster as a class, or to identify those objects that do not belong to any kind of noise cluster.

5) If the data flag is noise, the output is<Flag, Raw Data>, otherwise the output is <CID, (Flag, Raw_data)>.

**Step2(Local_DBSCAN_ReduceByKey):**

Input: <CID,list(Flag,Raw_data)>or<Flag,list(Raw_data)>

Output: <key, (centroid, ID)>

Process:

1) Reading raw data, and save partial cluster as new RDD through the method of saveAsTextFile.

2)Counting the number n of list(Flag,Raw_data) or list(Raw_data).

3) Calculating the average value of each field.

4)Building the centroid for each partial cluster.

5)The output is <key, (centroid, ID)>

In TABLE II, Flag means the attribute of each data; its values are 'core point', 'noise' and 'border'. CID is the cluster ID for each data, its initial value is key_1, and the updated CID when a new cluster is discovered.

### *C. merging the data partitions based on the centroid.*

After the above steps, each partition has produced independent clustering results. At this stage, we will merge the partial cluster to generate global clustering results. Since the distribution of each partition has similarities, so do the partial clusters. We design a merging strategy based on the centroid, its general idea is: Firstly, calculating the distance between every two partial clusters in the same partition; then use quick-sort or heap-sort to find the minimum distance $d_{\min}$ . Secondly, sort every $d_{\min}$ to find the minimum value $D_{\min}$ . Thirdly, setting the threshold $\sigma$ to merge partial clusters, and $\sigma \ll D_{\min}$ . Fourthly, creating a centroid_distance matrix and traversing every element in the matrix. If the distance is less than $\sigma$ , then add them to the merge queue until every element is visited. We require Combine_ReduceByKey, Combine_Reduce and ReLabel_Map to generate the partial cluster. The Schematic diagram of merging the data partitions based on the centroid is shown in Figure 5.

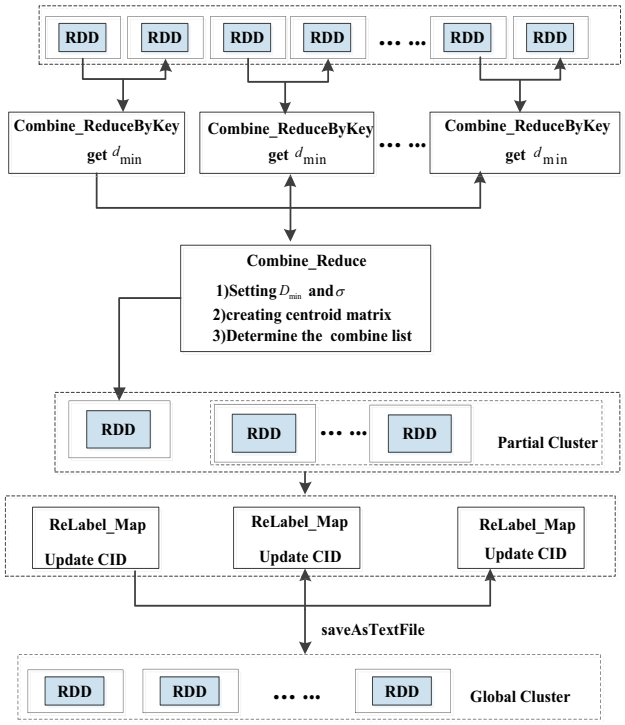


Figure 5. The Schematic diagram of merging the data partitions based on the centroid

The procedure of merging the data partitions is shown in TABLE III.

TABLE III. The procedure of merging the data partitions

**Step1(Combine_ReduceByKey):**

Input: <key, (centroid,CID)>

Output: <S, (list (centroid, CID), $d_{\min}$ )>

Process:

1) Reading raw data, get the list of centroid, and calculate the distance between every two partial cluster in the same partition.

2)Finding out the minimum distance $d_{\min}$ with quick-sort or heap-sort in the same partition.

3)The output is <S, (list (centroid, CID), $d_{\min}$ )>, and S is a unified identifier.

**Step2(Combine_Reduce):**

Input: <S, (list(centroid,CID), $d_{\min}$ )>

Output: <G_CID，list(combine-list)>.

Process:

1) Reading raw data, and save list (centroid, CID) and $d_{\min}$ for next stage.

2) sorting every $d_{\min}$ to find the minimum value $D_{\min}$ ,and setting the threshold $\sigma$

3)Creating centroid_matrix through list (centroid, CID).

4)Searching the centroid_matrix to find combine-list, the way of searching is shown in TABLE IV.

5)Getting the combine-list, the output is <G_CID，list(combine-list)>., and G_ID is global cluster identifier.

**Step3(ReLabel_Map):**

Input: <G_CID，list(combine-list)>，<CID, (Flag,Raw_data)>

Output: <G_CID, (Flag,Raw_data)>

Process:

1) Reading the raw data, and get the value.

2) Searching CID from combine-list, if succeed, update CID to G_CID, until every CID is updated.

3)The output is<G_CID, (Flag,Raw_data)>

The process of creating combine-list is shown in TABLE IV.

TABLE IV. The process of creating combine-list

Input: centroid_matrix, $\sigma$

Output: $list_1, list_2 .... list_n$

Process:

$Candidate = \{CID, Dis\}$

Make every element in centroid_matrix as unvisited

while(centroid_matrix!=null){

list=null

list.add( $CID_i$ )

centroid_matrix.Remove( $CID_i$ )

while(centroid_matrix!=null){

Candidate=find(list, centroid_matrix)

if(Candidate.Dis<= $\sigma$ ){

list.add(Candidate.CID)

centroid_matrix.Remove(Candidate.CID)}

else break}

end while

output(list)

end while}

## IV. SIMULATION

In this work, we set up the Spark cluster with similar computer machines. The cluster contains a Master node and several Slave nodes. Each compute node has 4G memory, Gigabit Ethernet, Inter Pentuim®Dual-Core ES5300 processor clocked at 2.6GHz, 500GB hard drive, Ubuntu14.04 64 bits operating system.

Application data comes from annual outpatient data of a certain disease, and has 225833 records, and 89 attributes. In this study, selecting 5 attributes about cost structure from the annual outpatient data as test data, to enable S_DBSCAN to solve practical problems. The detail of the test data is shown in TABLE V.

TABLE V. Detail of the test data

| Field | Type | Range |
|---|---|---|
| Accounting for inspection costs | Float | 0~1 |
| Accounting for drug costs | Float | 0~1 |
| Accounting for service costs | Float | 0~1 |
| Accounting for Medical costs | Float | 0~1 |
| Accounting for other costs | Float | 0~1 |

Using the text data to create four datasets for testing.D1 has 50 thousand records, D2 has 100 thousand records, D3 has 150 thousand records and D4 has 100 thousand records.

### A. Accuracy

In order to find out whether the S_DBSCAN algorithm could obtain the same results as the original DBSCAN algorithm use D1 as test data, and run DBSCAN algorithm and S_DBSCAN algorithm. The DBSCAN algorithm is performed by Weka[13], which is a common data mining tool. S_DBSCAN is performed on five computing nodes. Setting three groups of different parameters. The Comparative Result of the two algorithms is shown in TABLE VI.

TABLE VI. Comparative Result

| $Rate_1$ | $Rate_2$ |
|---|---|
| 3.15% | 0.92% |
| 2.91% | 1.12% |
| 2.32% | 0.98% |

$Rate_1$ is defined as follows:

$$Rate_1 = \frac{Noise(S\_DBSCAN) - Noise_{same}(S\_DBSCAN, DBSCAN)}{Noise(DBSCAN)} \quad (2)$$

where $Noise(S\_DBSCAN)$ denotes the number of noise computed by S_DBSCAN, $Noise(DBSCAN)$ indicates the number of noise computed by DBSCAN, $Noise_{same}(S\_DBSCAN, DBSCAN)$ is the same noise number computed by S_DBSCAN and DBSCAN algorithms respectively.

$Rate_2$ is defined as follows:

$$Rate_2 = \frac{Noise(DBSCAN) - Noise_{same}(S\_DBSCAN, DBSCAN)}{Noise(DBSCAN)} \quad (3)$$

It can be seen from TABLE VI, S_DBSCAN algorithm has a good clustering result, and a lower error rate. S_DBSCAN algorithm can obtain almost the same results as traditional DBSCAN algorithm.

### B. Running time

Running time is used to determine the speed of execution of the algorithm. Running time can be used to evaluate whether S_DBSCAN algorithm improves the operating efficiency. The experiment is based on a conventional stand-alone platform and a distributed Spark cluster. The test data are D1, D2, D3 and D4. For each set of tests run multiple times, we take the average running time as the final result. The running time results are shown in Figure 6.

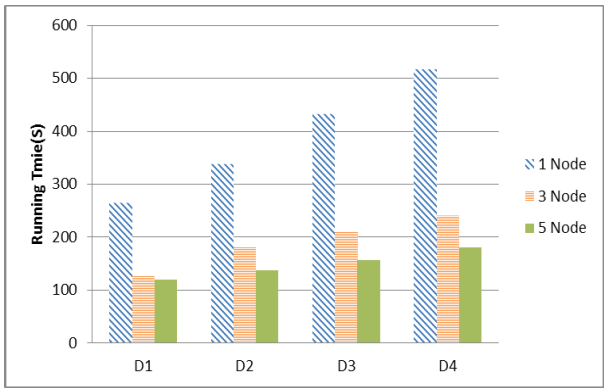


Figure 6. Running time of S_DBSCAN algorithm

Seen from Figure 6, there is a significant difference between running in a single-node or multiple-node utilization. With the increasing size of the dataset, the difference between the number of utilized nodes is growing. For the same data set, the running time decreases with the increase of computing nodes, thus reflecting the parallel ability of S_DBSCAN algorithm.

### C. Speedup

Speedup refers to the ratio of single-task processing time as compared to multiple-task processing time for the same task. Speedup is usually used to measure a parallel system, a program parallelization performance, or results. In order to test the parallel ability of the S_DBSCAN algorithm, we use D1, D2, D3 and D4 to test the S_DBSCAN algorithm. The speedup results of S_DBSCAN algorithm is shown in Figure 7.

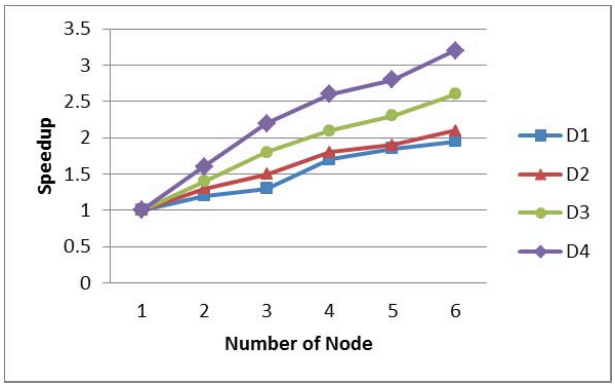


Figure 7. The speedup results of S_DBSCAN algorithm

Seen from Figure 7, the S_DBSCAN has significant speedup. For the same set of data, the more nodes utilized, then the greater the speedup. For the same number of nodes, the larger the data set, the greater the speedup. This fully reflects the superior efficiency of the S_DBSCAN algorithm when dealing with massive data, as compared to existing parallel DBSCAN algorithms.

## V. Conclusion

To solve the low efficiency problem in data partitioning and partition merging for the existing parallel DBSCAN algorithms, this paper proposes a parallel DBSCAN algorithm based on Spark named S_DBSCAN; which could quickly realize the merges and divisions of clustering results from the original data. This paper evaluates the S_DBSCAN algorithm by dealing with annual outpatient data. The experimental result shows the proposed S_DBSCAN algorithm can effectively; and efficiently; generate clusters and identify noise data. In short, the S_DBSCAN algorithm has superior performance when dealing with massive data, as compared to existing parallel DBSCAN algorithms.

## References


[1] Fayyad U M, Piatetsky-Shapiro G, Smyth P, et al. Advances in knowledge discovery and data mining[J]. 1996.

[2] 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.

[3] Xu X, Jäger J, Kriegel H P. A fast parallel clustering algorithm for large spatial databases[M]//High Performance Data Mining. Springer US, 1999: 263-290.

[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.

[5] Welton B, Samanas E, Miller B P. Mr. scan: Extreme scale density-based clustering using a tree-based network of gpgpu nodes[C]//Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. ACM, 2013: 84.

[6] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[7] B.-R. Dai and I.-C. Lin, "Efficient map/reduce-based dbscan algorithm with optimized data partition," in Cloud Computing (CLOUD), 2012IEEE 5th International Conference on. IEEE, 2012, pp. 59–66.

[8] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan, "Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce," in Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on. IEEE, 2011, pp. 473–480.

[9] Apache Spark[OL].http://spark.apache.org/,2014

[10] Shoro A G, Soomro T R. Big Data Analysis: Apache Spark Perspective[J]. Global Journal of Computer Science and Technology, 2015, 15(1).

[11] Qi Y U, Jie L. Research of cloud storage security technology based on HDFS [J][J]. Computer Engineering and Design, 2013, 8: 013.

[12] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C] //Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2.

[13] Hall M, Frank E, Holmes G, et al. The WEKA data mining software: an update[J]. ACM SIGKDD explorations newsletter, 2009, 11(1): 10-18.