I.J. Information Technology and Computer Science, **2022, 6, 1-12**
Published Online on December 8, 2022 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2022.06.01

Parallel DBSCAN Clustering Algorithm Using Hadoop Map-reduce Framework for Spatial Data

Maithri. C.*
Department of Computer Science and Engineering, Kalpataru Institute of Technology, Tiptur, India
E-mail: maithri.c.prashanth@gmail.com
ORCID iD: https://orcid.org/0000-0003-0300-3269
*Corresponding Author

Chandramouli H.
Department of Computer Science and Engineering, East Point College of Engineering and Technology, Bangalore, India
E-mail: hcmcool123@gmail.com
ORCID iD: https://orcid.org/0000-0003-0453-6358

Received: 10 July 2022; Revised: 14 September 2022; Accepted: 14 October 2022; Published: 08 December 2022

Abstract: Data clustering is the first step for future applications of big data analysis. It is a driving model for Artificial Intelligence and Machine Learning architectures. Processing large volumes of data in faster mode is a big challenge in these applications. which requires fast and efficient algorithms for handling big data. Parallel clustering algorithms are one promising design, which increases the speed of handling such big data. In this paper, a parallel algorithm for clustering a spatial dataset called the P-DBSCAN algorithm is implemented using Hadoop map-reduce framework. This research paper signifies the improvement for data clustering in data analytic applications. The new P-DBSCAN algorithm is executed over generated dataset. The result of this parallel algorithm is compared with existing DBSCAN algorithm to show improvement of runtime performance. This work offers an increase in the performance of execution time. In addition, the outcome of P-DBSCAN shows how to resolve the scalability problem of a large data set.

Index Terms: Artificial Intelligence, Data mining, DBSCAN, Hadoop, Parallel Clustering.

1. Introduction

In machine learning techniques, clustering the large data files and performing an analysis on clustered data is one of the primary methods. This data mining technique describes a process of dividing large data objects into disjoint subgroups where each subgroup will contain similar data objects. This process of partitioning method to obtain a subgroup of similar data is called clustering. A clustering algorithm performs the discovery of groups from given data by studying similarity property in-group of items and reducing dissimilarity in individual clusters. Clustering algorithms are all unsupervised methods where a process of grouping will not have any pre-knowledge of any status of data. Clustering algorithms are broadly classified into four types: partitioning-based, hierarchy-based, density-based, and grid-based. To achieve the improvement in the data clustering process parallel algorithms are implemented. The major objective of this research paper is to design an approach to optimize result computation of DBSCAN through using a parallel implementation. Parallel computation of DBSCAN will increase computational efficiency and data representation.

The parallel implementation for clustering algorithms is very few in practical use, and considering spatial data there is very little knowledge for parallel computation [1, 2]. Thus, addressing improvement for the performance of clustering algorithms in spatial access methods to formulate clusters is an issue. With this consideration, we implemented and studied a parallel approach for DBSCAN clustering by considering performance improvement for execution on large datasets. The focus of this research paper is to lower computation time, but not changing the original DBSCAN algorithmic approach. This paper aims to utilize parallel software engineering and parallel programming environments using high-level structural programming languages to make a productive code for spatial data clustering.

Through this paper we are showing the parallel implementation of DBSCAN algorithm over the Hadoop Map-Reduce framework model to increase the computation of forming the clusters.

2. Related Works

Clustering algorithms are all unsupervised learning methods for dividing data points into several similar groups based on similar properties or conditions. Today's machine learning platforms require a clustering process to analyze data in a proper and faster manner. There are many algorithms got defined and classified in the field of clustering, such as K-Means (based on a distance between points) [3], DBSCAN (based on a distance between nearest points) [4], Spectral clustering (based on graph distance), Affinity propagation (based on graph distance) [5], Mean-shift (based on a distance between points) [6] etc.

One of the most searched and a used clustering technique is the DBSCAN method. DBSCAN [4, 7] is an approach for modeling clustering that is used in a grouping of data to form clusters with areas of higher density to lower density. Data points that are not included in a cluster are noise points. One of the DBSCAN algorithm's main features, compared to another algorithm, is that it detects arbitrary shaped clusters of data points and the algorithm does not require any previous information of clusters as defined in K-Means and others [8, 9]. The DBSCAN algorithm consider only two input parameters (threshold and minpts) to identify cluster area that will have dense of data points. The algorithm will run reasonably faster as it reads a linear number of data values. The worst-case runtime complexity of DBSCAN is $O(n^2)$. Using proper indexing structures such as KD-tree or R-tree, it is possible to improve the runtime complexity of an algorithm. It is claimed in practice that the performance of DBSCAN can be improved up to O(nlogn) with the proper inclusion of indexing data structures. From past research, many improvements are defined to DBSCAN to have a better performance. I-DBSCAN [10] shows improvement for sampling data points by reducing the number of range computations based on large spatial database information.

The new enhanced approach for the DBSCAN is described in Liu et al. [11], which introduced as VDBSCAN is defined to overcome a problem in finding clusters with different densities on a single dataset. This algorithm doesn't require input parameters to be determined by the user. The VDBSCAN will automatically choose arbitrary values for parameters. The algorithm starts with the first steps for selecting epsilon and diverse density values for clusters. This algorithm is having the same time complexity as the traditional DBSCAN algorithm. F-DBSCAN [11, 12] defines a faster range computation for a sampling of input data points.

The other set of algorithms, such as GF-DBSCAN [13] and GRID-DBSCAN [14] uses partitioning of data to formulate a grid of data points. Theoretically, these algorithms have not made any changes to the original DBSCAN. Thus, the running time of GRID-DBSCAN is as same as the original DBSCAN algorithm, except the algorithm goes with grid arrangement on input data points.

In the paper [15], Zhou proposed an algorithm called FDBSCAN that makes faster computation than the DBSCAN algorithm; thus, runtime complexity is improved. FDBSCAN study considers only a few objects for core object neighbor as a starting point and increases this group to form one cluster.

In the paper, [16] Roy and Bhattacharyya described a new algorithm called EnDBSCAN. This algorithm is a combined approach with the working of optics over DBSCAN. The algorithm detects and forms cluster structure over given data with noise. This approach is using the distance function of the optics algorithm and starts finding core points for clusters. Results obtained from EnDBSCAN showed detection of embedded and nested clusters but have a problem in runtime complexity. This algorithm also requires only a few obstacles as compared to the OPTICS algorithm.

The study of Viswanath et al [17] in their paper made a new proposal L-DBSCAN algorithm, which is a hybrid clustering for overcoming runtime complexity. This algorithm is applied using a large dataset and also to get arbitrary-shaped clusters. The algorithm is derived using two-level of prototypes and requires two parameters that the user gives. The experimental result of L-DBSCAN is applied only on a suitable dataset with user-defined parameters and shown less time needed for a finding of arbitrary shaped clusters.

Birant et al. [18] in their proposal discussed a new method called ST-DBSCAN that can detect spatial and non-spatial clusters using attributes of the dataset. This algorithm takes three parameters to detect and formulate clusters. Many applications use this approach, such as geographic information systems, medical imaging, and weather forecasting. The outcome result of this algorithm has given a promising solution for the detection of spatial-temporal data clusters.

A study carried out by Tamura et al. [19] has defined a DBSCAN clustering method to work over both spatial and temporal data from large geo-referenced documents. This algorithm used latitude, longitude, time interval, and a minimum threshold value of document as values of parameters for recognizing both temporal and spatial clusters. They used 480,000 tweets or more datasets and evaluated the algorithm. The experiments of Nitta et al. [20] described a new method for real-time event detection. This algorithm is using latitude, longitude, time information, and text tag information of images. This algorithm has two-step event detection processes. The experimental result gives increase efficiency in detecting clusters for a small amount of image data.

The work of Popovici et al. [21] is based on an online clustering approach that has extended DBSCAN using evolving data streams. This algorithm takes four parameters, and evaluation result shown a positive result for cluster formulation.

The work of Götz et.al [22] shows the implementation of a parallel algorithm for DBSCAN as HPDBSCAN using the parallel architecture of Spark. The evaluation result shows the improvement in speed and scaling of the dataset.

The implementation of parallel DBSCAN of Szénási, S. [23] for GPS coordinates of accidents shows the

parallelization of DBSCAN approach. The results of CPU and GPU execution show the accuracy of the parallel method.

The work of M. Chen et al [24] resolves the bottleneck of the DBSCAN algorithm. It is a novel approach in a distributed environment. The parallel implementation of DBSCAN explained by Ling et al, [25] uses partition of data and construction of R-tree. This algorithm is implemented using Spark and python.

G. Luo et al [26] shown a parallel implementation of DBSCAN called S-DBSCAN, which creates data partitions to increase computational speed. The algorithm is implemented on Spark distributed system.

3. Preliminary Concepts

DBSCAN is one of the sequentially processing algorithms for formulating clusters and noise of given spatial data. Sometimes, it is used over the temporal data clustering process. The original paper [13] provides a complete representation of the algorithm briefed in the following section. By density-based, it is meant that data points in a region are connected with certain conditions forming a dense cluster. The cluster is formed as a set of data points that are interconnected with each other based on certain conditions. The algorithm uses a threshold value called minPts, which is used as a condition. If a data point is within threshold density, then that data point is in a cluster. Density is defined as spaces where data points are densely connected with a distance threshold. If density falls below a given threshold, data are regarded as noise. It also defines the distance between neighbor data point with epsilon (ε) value. Consider that N is given a set of data points, then the following definitions are defined in the DBSCAN procedure.

Definition 1: For a data point p, the set of neighbor data points is denoted as $N_\varepsilon(p)$, is defined by

$$N_\varepsilon(p) = \{q \in N | dist(p,q) \leq \varepsilon\} \quad (1)$$

where, N is a given data points and dist(p, q) is any distance function for example Jaccard function.

Definition 2: A Point p is a core point if and only if $|N_\varepsilon(p)| \geq$ minPts.
Definition 3: Data point q is said to be density reachable from data point p if and only if $q \in N_\varepsilon(p)$ and p is a core point.
Definition 4: A data point q is a border point if and only if q is density-reachable from core point p and $|N_\varepsilon(p)| <$ minPts.
Definition 5: A point q is density-reachable from a point p if and only if there is a chain of points $q_1 .. q_n$ with q_1=p and q_n=q such that q_i+1 is directly density-reachable from q_i.
Definition 6: If there exists a data point r such that p and q are density-reachable from r, then point q is a density-connected point from point p.
Definition 7: A cluster C is a non-empty subset of N, where N is a set of points that must satisfy the following conditions:

1. ∀p, q points if p belongs to C and q is density-reachable from p, then q ∈ C.
2. ∀p, q ∈ C then each point p is density-connected to q with respect to ε and minPts.

Definition 8: A point is a noise point when it is not a core point nor a border point. Noise points are in N but, it is not in any of the clusters.

Procedure for computing cluster of data points from given data points using DBSCAN approach is shown in algorithm 1. Here N is input data points, with threshold ε, and density value minPts.

Algorithm 1: DBSCAN (N, ε, minPts)

1. Set cluster id value G = 0
2. for each point a ∈ N that is not classified to cluster do
    a. $N_\varepsilon(a)$ = CorePoint(a, ε)
    b. if $|N_\varepsilon(a)| \geq$ minPts then
        i. Set a's cluster id to G
        ii. ADDCluster($N_\varepsilon(a)$, G, ε, minPts)
        iii. G ← G + 1
    c. else
    d. Label a as noise

Procedure CorePoint will compute neighbors points for point a with threshold ε. The computation will go based on definition 1.

Algorithm 2 is a sub procedure used to expand clusters for density reachable points. N is a set of neighbor points of p.

Algorithm 2: ADDCluster(N, C, ε, mPts)

1. for each p ∈ N do
2. if p is not in cluster C then
    a. $N_\varepsilon(p)$ = CorePoint(p, ε)
    b. if $|N_\varepsilon(p)| \geq$ mPts then
    c. $N = N \cup N_\varepsilon(p)$
    d. if p is not added to any cluster then
        i. form a new cluster with cluster-id from p cluster-id C

For any point a, the neighbor Nε(a) is computed as

$$N\varepsilon(a) = \{b \in N \mid dist(a,b) \leq \varepsilon\} \quad (2)$$

The runtime complexity of the original DBSCAN is $O(n^2)$. But, the runtime of an algorithm can be improved efficiently to O(nlogn) using parallel computation models such as KD-trees or R-trees for identifying core points. Another fact that is noticed in DBSCAN is that identification of border data points is non-deterministic in which a border point may be directly reachable from more than one data point which may be present in more than one cluster. Thus, this data point is added to a first occurred cluster which is a non-decision-based condition. DBSCAN will start its computation by selecting an arbitrary point, thus, the formulation of clusters using this algorithm will vary with each execution. There are many improvements defined to fine-tune clusters, increase performance, and handle large inputs for computation. The complete algorithm efficiency is depending on the data structure used for execution.

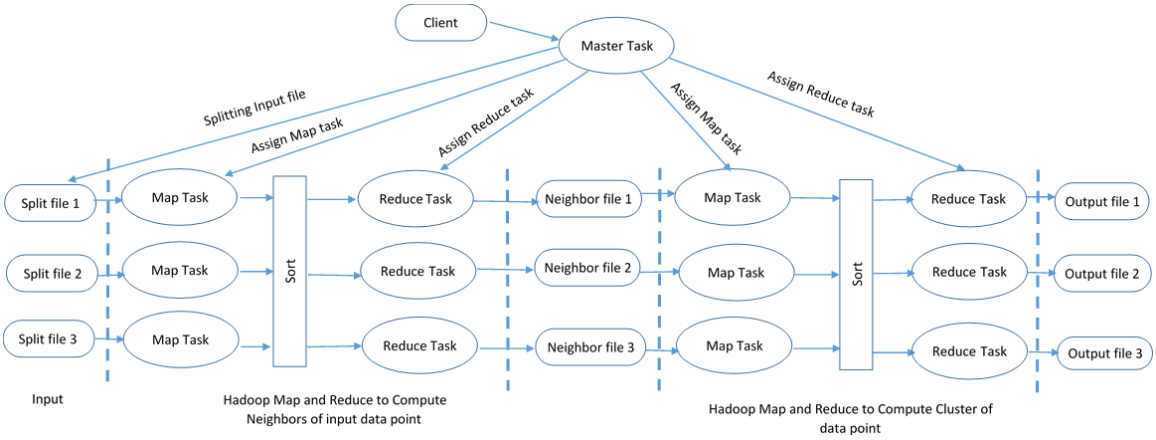


Fig.1. System overview of P-DBSCAN.

4. P-DBSCAN Algorithm

P-DBSCAN is a parallel implementation for the basic DBSCAN algorithm defined on a map-reduce framework of the Hadoop distributed system. Mapper and reducer programs are designed in such a way so that input data points will get into the cluster formulation. There are three steps involved in the mapper algorithm. The first step is partitioning data into grid form. The second step is determining core points by calculating K-neighbors. The third step is to merge density-connected core points as a cluster and to identify noise data points. Fig.1 shows a system overview of P-DBSCAN implementation. The system model of P-DBSCAN has many mapper and reducer tasks that runs on the data nodes of the Hadoop. Each mapper task reads part of dataset performs the task of P-DBSCAN. The algorithms are defined below.

4.1. Partition of Data Points

The mapper function will start with a partitioning of given data points into a grid. Grid structure goes with cell dimension and will have a diagonal length of ε, so the width of a grid cell is ε/√2. Given data points are placed into grid cell based on x and y values. The decision of grid cell width ε/√2 will make the cell contain more than minPts and gives the ability to mark all cell points as core points.

Fig.2 shows the grid construction. It is true in this representation that for maximum distance between data point's p and q in a grid cell is ε. The number of points within a grid cell is more than minPts, then, it is true that the cell will be at least equivalent to minPts. To grid, an extra row and column are added to calculate edge data point computation accurately. Algorithm 3 will construct partitions with runtime complexity equal to O ($n + n_{Cells}$) time in the worst case.

For proof, the runtime for algorithm 3 is shown below. The line 2 process can be done with linear time, just simply searching N. Similarly, for loop present in line 6 is also executed in linear time and the runtime for line 5 is O (n_{Cells}).

Thus, the total runtime for this algorithm is O($n + n_{Cells}$) in the worst case. With the small values of ε, a grid can be of very large sparse data with large maxX − minX and maxY − minY, n_{Cells}. This makes algorithm 3 to be less efficient.

To avoid efficiency, the number of cells will be limited to at most O(n).

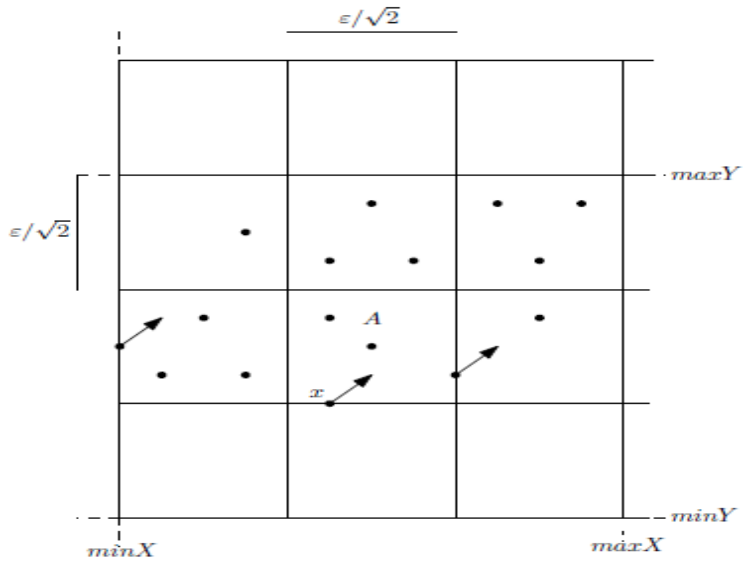


Fig.2. Showing grid construction with data.

Algorithm 3 BuildGrid(N, ε,)

1. cW ← abs(ε/√ 2)
2. Set border values of Grid as min_X, max_X, min_Y, and max_Y from N
3. Total rows is nRows ← floor((max_X − min_X)/ cW +1)
4. Total columns is nCols ← floor((max_Y − min_Y)/ cW + 1)
5. Set grid G as an empty grid and G size is nRows*nCols.
6. for each data point p from N do
    a. Add point p to G as
    [floor((p.x-min_X)/cW+1)]
    [floor((p.y-min_Y)/cW+1)]
7. return G

The Number of rows and columns for algorithm 3 is computed as

$$\text{rows} = \left\lfloor \frac{maxX - minX}{cW} + 1 \right\rfloor \tag{3}$$

$$\text{cols} = \left\lfloor \frac{maxY - minY}{cW} + 1 \right\rfloor \tag{4}$$

Hash tables are used for solutions happening with a regular grid which can have many empty cells. Key for point p is given as

$$\text{p. key } = \left\lfloor \frac{p.x - minX}{cW} + 1 \right\rfloor * (\text{ncols} + 1) + \left\lfloor \frac{p.y - minY}{cW} + 1 \right\rfloor \tag{5}$$

Note that if point p is in G[i, j] then it is at a distance i · (nCols + 1) + j.

Hash table H[0..n] is initialized with a hash function $h: \{0..n_{GridCells}\} -> \{0..n\}$. A hash function maps key points before it enters a location in memory where cell points are stored. Using an optimized hash function will make search and inserting of value in O(1) time.

4.2. Sorting-based Algorithm

Another approach to creating a non-regular grid is using a sorting algorithm over data points in x-coordinate and then sorting data points in the y-coordinate form. The cell width in this case also will be ε/√ 2. The first cell will be of height ε/√ 2 which forms a box of data points. Now, the process will go for finding the next data point which can make another box. This process is continued until no more data point is left. The approach is defined as a sorting-based algorithm, as shown in algorithm 4. An algorithm can be executed in a parallel way since data points are all in sorted order. Using this algorithm, we can compute boundary information of each box and store this computed boundary information in memory. Fig.3 shows a sample for the formulation of boxes and adding data points to boxes.

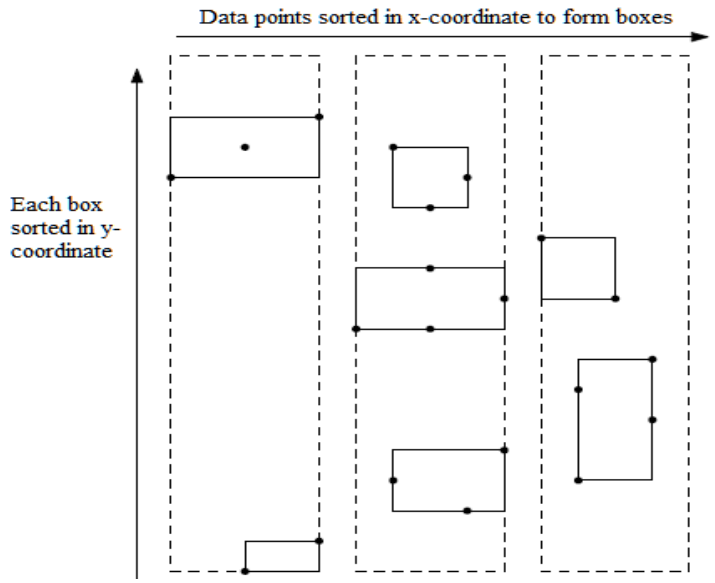


Fig.3. Formulation of non-regular grid using sorting based algorithm.

Algorithm 4 BuildBoxes(N, ε)

1. cW ← ε/√ 2
2. Set G to number of boxes in grid as empty
3. Sort data points N in x-coordinate order
4. Set S as empty set of points.
5. Set S[1]-> first point p_1 from N
6. for i ← 2 to |N| do
    a. q ← S[1]
    b. if p_i.x > q.x + cW then
        i. AddBoxToGrid(G, S, cW)
7. Remove all points from S
8. Add last point p_i to S
9. AddBoxToGrid(G, S, cW)
10. return G

Algorithm 5 AddBoxToGrid(G, S, cW)

1. Sort the set S in y-axis
2. Set B as empty set
3. B[1]=S[1] add first point to B from S
4. for i ← 2 to |S| do
    a. q ←S[i]
    b. if q.y > B[1].y + cW then
        i. Add B to G
    c. Delete all points from B
    d. B[1]=q
    e. Add B to G

Total runtime for algorithm 4 defined is:

$$O\ (n \log n) + O\left(\sum_{i=1}^{|S|} |Si|\ log|Si|\right)$$

$$\leq\ O(n \log n) + O\left(\sum_{i=1}^{|S|} |Si|\ log\, n\right)$$

$$\leq O(n \log n) + O\left(\log n . \sum_{i=1}^{|S|} |Si|\right)$$

$$\leq O(n \log n) + O(\log n \cdot n)$$

$$\leq O(n \log n)$$

4.3. Finding Core Points

This is the next step to the mapper process, which will iterate through non-empty cells after partitioning algorithm execution. Algorithm 3 may generate some empty cells in a grid. So it is necessary to create a list of all non-empty cells before identifying core points. To create non-empty cells is a simple step of creating a new list that contains only cells with data points which happens by scanning a list of cells and picking only those cells whose length is greater than 0. This process will take that is equivalent to algorithm 3. The process of finding core points within grid cell will start with finding grid cell points > minPts. If so, that point is marked as a core point. If not so, we check each point inside that cell to determine other points as core points. To identify core point p in a cell, check neighboring cell points $N_\varepsilon(p)$. Still, the algorithm takes O(1) to check boxes of data points. The next step is to compute each data point's distance in cells that are neighbors for p. Process terminates when all data points inside a cell are checked off if we found minPts data points that are having distance ≤ ε. Algorithm 6 shows this procedure in which |A| is several data points in grid cell A. G is nonempty cells means grid cell with data points.

Algorithm 6 FindCorePoints(G, ε, minPts)

1. for each A∈ G do
2. if |A| > minPts then
    a. set all points p in A as core point.
3. else
    a. for each point p ∈ A do
        1. set n_{points} as 0
        2. for each cell B ∈ Nε(A) do
        3. for each point q ∈ B do
            1. if dist(p, q) ≤ ε then
            2. increment n_{points} by 1
            3. if $n_{points} \geq$ minPts then
            4. set p as core point
            5. break
            6. if $n_{points} \geq$ minPts then break

4.4. Merging Clusters

Merging clusters happen based on a distance between two core points if it is at most ε. This is a fact that two points are belonging to the same cluster. Example in fig.3 shows two clusters with eight points which shows that core point a is also belonging to the same cluster. If the distance is greater than ε then the cluster is formed as four points as shown in fig.4. These cells are considered nodes present in the same cluster. To elaborate the point of merging, an edge is added for cells A and A' only if there is a q and q' present in A and A' and distance dist (q, q')≤ ε.

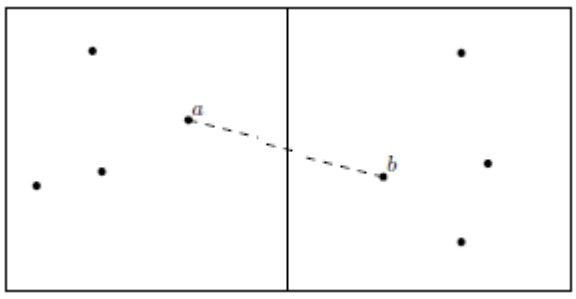


Fig.4. Merging of clusters.

4.5. Finding the Border and Noise Points

The final step of the mapper is to identify border and noise points. This is done using algorithm 7 which is executed over a set of data points N, not core points. Points in N may be the border point of a cluster if there is at least one core point in a neighbor of that point Nε(a). The rest of all other points are identified as noise points by an algorithm.

Algorithm 7 FindBorderPoint(N, ε)

1. for each A ∈ N do
2. for each data point a in A do
    a. if a is not core point then
    b. set b to null
    c. for each grid cell that is neighbor of A, $N_\varepsilon(A)$ do
        i. temp ← FindNearerCorePoint(p, A)
        ii. if dist(a, temp) ≤ dist(a, b) then
            1. b ← temp
            2. if b = =null then
                add point a to cluster of b
            3. else
                set point a as noise

Algorithm 7 as runtime complexity O (n. minPts) that can be proved similarly as defined for algorithm 6. Differences between algorithm 6 and algorithm 7, that algorithm 7 will check only neighbor points for core points instead of checking all points. Thus, algorithm 7 is more efficient than algorithm 6.

5. Experiments

An experimental setup is created to check two implementations of the algorithm using the Hadoop distributed file system. Worst-case time for traditional DBSCAN algorithm $O(n^2)$. The new algorithm defined as P-DBSCAN will have the worst time complexity as O(n log n). Experimental results of the new algorithm P-DBSCAN are compared to DBSCAN. The first implementation has resulted in $O(n^2)$. The second, which contains grid formulation, has runtime equivalent to $O(n*n_{cells})$, which is slightly more efficient than the original algorithm. Thus, a new algorithm is similar to DBSCAN with the inclusion of grid arrangement for data to optimize search of core point. Grid-based range search is applied on search procedure based on distance ε, so only O(1) cells are visited. The new algorithm will run in O(n) in practice if we have O(1) of points inside a cell in the best case. But, the worst-case runtime is $O(n^2)$. Implementation is done using Java 1.8 programming with NetBeans 12 as IDE.

A Hadoop 3.1.0 is set up with one master node and three slave nodes. The master node is installed on the HP ProLiant G580 server with 4 processors of Xeon 2.93GHz processor having 4 cores. A total of 16 cores processor architecture is used to run implementation. The server system has 128GB RAM and 480 GB of Storage. The master node runs the application over three slave nodes. The slave node configuration is Intel Core i3 CPU 2.53GHz, 4GB RAM, and 500GB Storage. Both master and slave nodes are running on Ubuntu 18.04 operating system.

Fig.5 shows the Java GUI interface developed for the generation of a dataset and to execute the P-DBSCAN algorithm over Hadoop.

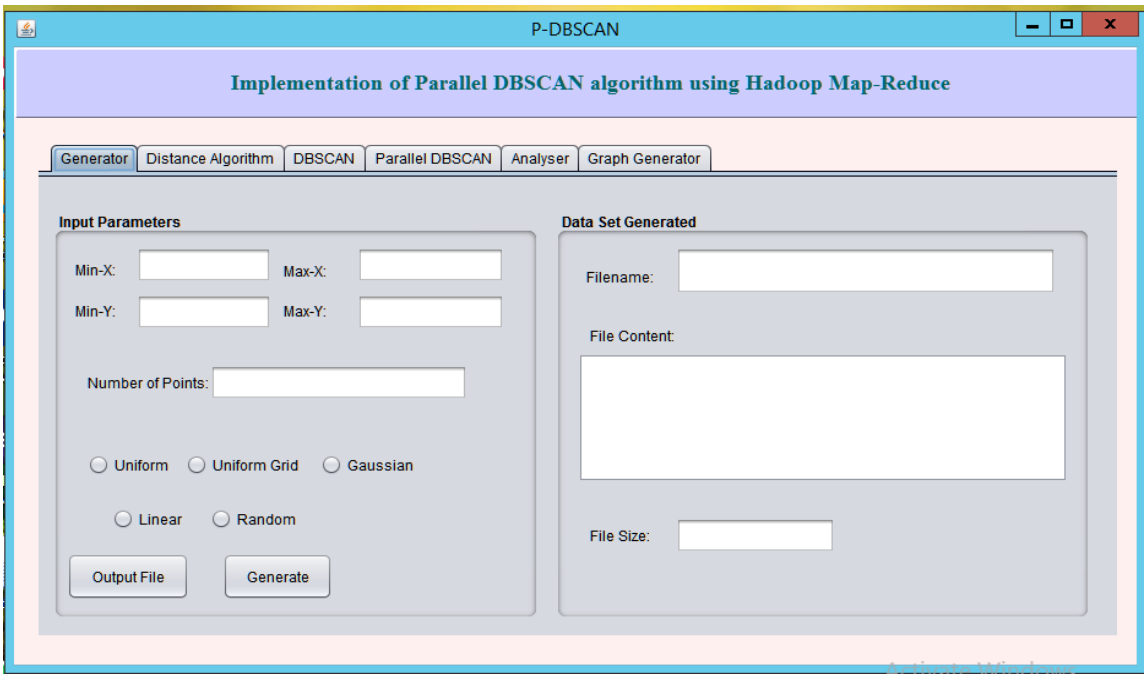
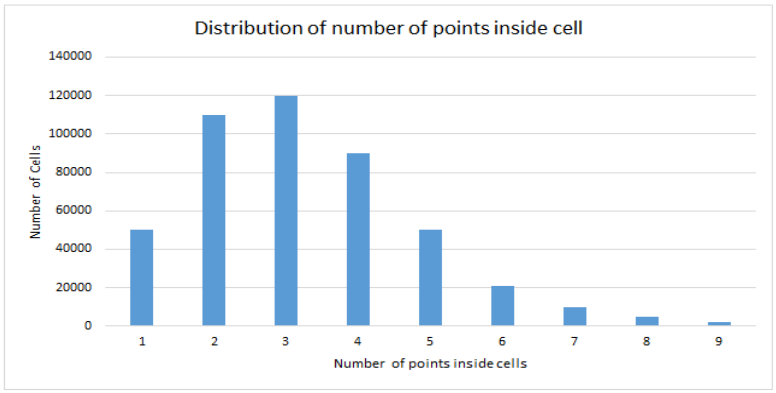


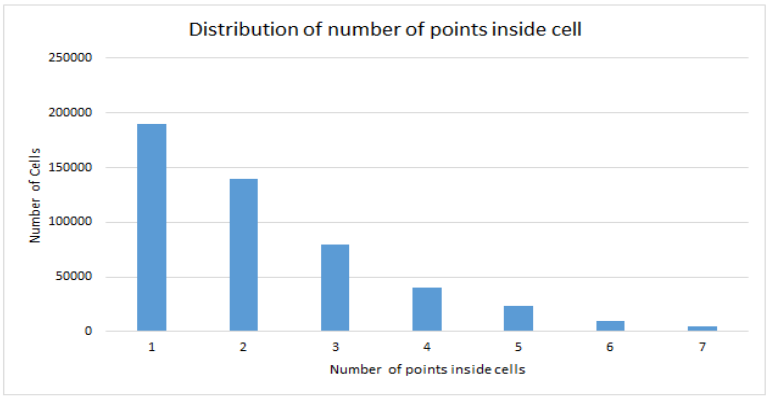
Fig.5. Experimental setup for executing P-DBSCAN algorithm.

5.1. Dataset

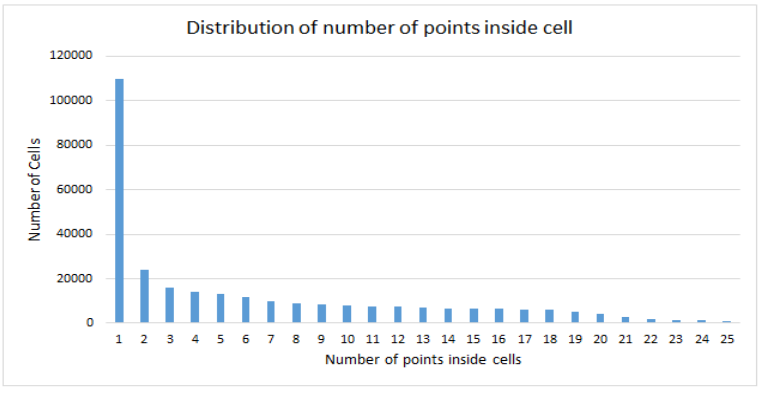
For the experimental purpose, a data point generator is designed which will generate a dataset based on input conditions. Three types of dataset are generated as shown in fig.6 for evaluation purposes.



Uniformly Random Data Points



Uniform Boxes



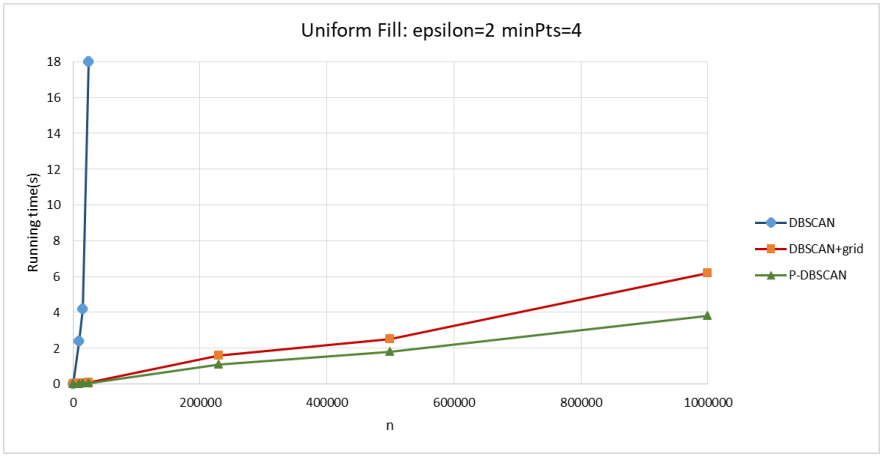
Gaussian Distribution

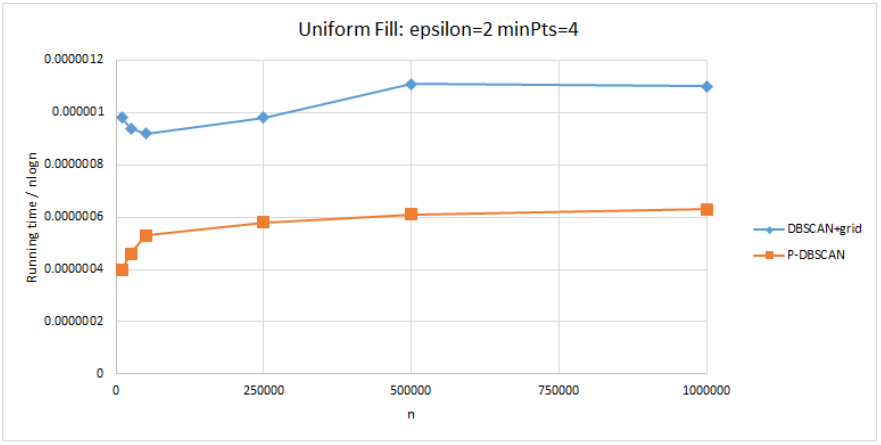
Fig.6. Various data sets generated for experimental purpose.

1. Uniformly random data points inside bounding box $\sqrt{n} \times \sqrt{n}$. The range of data used in this case is n=1000000 and $\varepsilon = 2$. This generation gives an equal distribution of data points in a cell has shown in fig.6. Looking at the data shown in the chart there are very few empty cells defined. Data points inside the cell will vary from 12 to 15 at maximum. Only a few cells will contain this range of values.

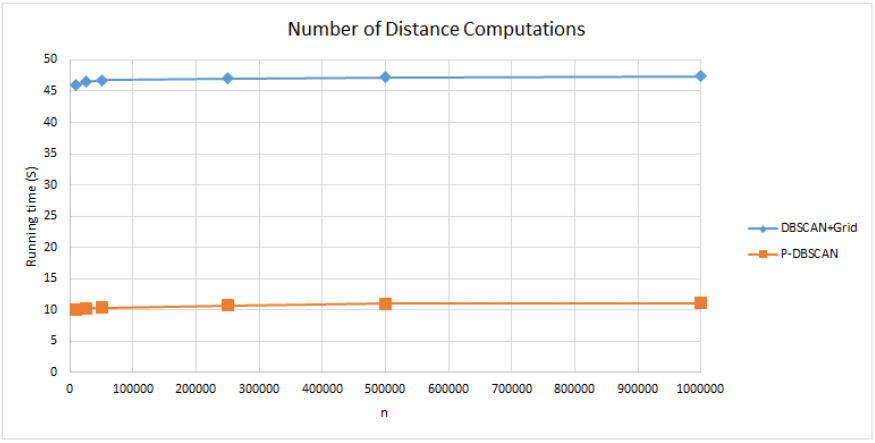
2. Uniform Boxes, in this case, data points generated will be m, equal to 0.9*n of total points. Datasets are generated uniformly, which contain m boxes. For experimental purposes, we have considered the value m=5 for a number of boxes. Here it is considered that n-m points are noise. Box size will be bounded by $5\sqrt{m} \times 5\sqrt{m}$, with each box has a diameter $\sqrt{m}$. The experimental setup goes with n = 100000, and $\varepsilon = 4$ is the dataset generated.

3. Gaussian distribution of data points generated is the same as uniform boxes with additional condition mean= center of the box and with standard deviation as 1/6.





Running time / nlogn



Distance computation over n

Fig.7. Experimental result obtained using ε and minPts fixed.

The original DBSCAN algorithm and new P-DBSCAN algorithm implemented is executed over data set size from n=10000 to n=1000000 with setting epsilon and minPts to a fixed value. Sample results for execution time versus several data points are charted in fig.7. The result shows a comparative performance of improving the P-DBSCAN algorithm over DBSCAN and DBSCAN with grid approach. Comparison of proposed method with existing algorithms demonstrated in table 1.

Table 1. Running time comparison of the proposed model.

Running Time	**DBSCAN method**				**Proposed Model**
	Optimized	Rtree-ReducedBounder	Rtree-Cost based	DBSCAN+grid	P-DBSCAN
time(s)	4ms	4ms	4ms	6.2ms	3.78ms

6. Conclusions

Data clustering is now an essential module in the field of big data analytics and machine learning techniques. Implementation of parallel algorithms for increasing the efficiency of the computation is a need in modern algorithmic designs. This research paper work demonstrated a new algorithm P-DBSCAN implementation over the Hadoop Map-Reduce framework environment to achieve runtime efficiency. Experimental results obtained with clustering of n with different forms of dataset shows that the P-DBSCAN algorithm runs efficiently. The table 1 projected the running time of execution, which shows the P-DBSCAN algorithm takes less time than the DBSCAN algorithm.

Detail analysis is performed on a new algorithm using map-reduce implementation over Hadoop-distributed system. A new algorithm has better and faster execution with selected values of epsilon and minPts.

For further research, it can be carried to improve the new algorithm by considering certain facts that, grid cells can be analyzed in a merged manner with several data points with less than minPts. A merged grid will not exceed minPts condition. Another improvement can be performed on the sorting algorithm so that further optimization of algorithm can be achieved. Adapting algorithm on higher-dimensional space also an improvement for the P-DBSCAN algorithm.

References

[1] M. Chen, X. Gao and H. Li, "Parallel DBSCAN with Priority R-tree," 2nd IEEE International Conference on Information Management and Engineering, pp. 508-511, 2010, doi: 10.1109/ICIME.2010.5477926.

[2] A. K. Jain, M. N. Murthy, and P. J. Flynn. "Data clustering: a review." ACM Computing Surv., pp. 264–323,1999, doi: https://doi.org/10.1145/331499.331504.

[3] Zhao W., Ma H., He Q. "Parallel K-Means Clustering Based on MapReduce." In: Jaatun M.G., Zhao G., Rong C. (eds) Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science, vol 5931. Springer, Berlin, Heidelberg. 2009, doi: https://doi.org/10.1007/978-3-642-10665-1_71

[4] Martin Ester, Hans-Peter Kriegel, J¨org Sander, and Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996, https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

[5] Robson L. F. Cordeiro, Caetano Traina Jr. "Clustering Very Large Multi-Dimensional Datasets with Map Reduce." 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 690-698, 2011, doi: https://doi.org/10.1145/2020408.2020516

[6] Jin, R., Hong, L., Wang, C., Wu, L., & Si, W., "A Hierarchical clustering community algorithm which missed the signal in the process of transmission," Review of Computer Engineering Studies, pp. 27-34, 2015, http://www.iieta.org/sites/default/files/Journals/RCES/02.3_06.pdf.

[7] B J Frey and D Duech, "Clustering by passing messages between data points," Technical Report in Science Journal, Vol. 315, Issue 5814, pp. 972-976, 2007, doi: 10.1126/science.1136800

[8] Von Luxburg, U. "A tutorial on spectral clustering", Statistics and Computing 17, pp.395–416, 2007. https://doi.org/10.1007/s11222-007-9033-z.

[9] K. Govindarajan, T. S. Somasundaram, V. S. Kumar and Kinshuk, "Continuous Clustering in Big Data Learning Analytics," IEEE Fifth International Conference on Technology for Education, Kharagpur, pp. 61-64, 2013, doi: 10.1109/T4E.2013.23.

[10] B. Borah and D. K. Bhattacharyya, "An improved sampling-based DBSCAN for large spatial databases," International Conference on Intelligent Sensing and Information Processing, Chennai, India, pp. 92-96, 2004, doi: 10.1109/ICISIP.2004.1287631.

[11] P. Liu, D. Zhou and N. Wu. "VDBSCAN: Varied Density Based Spatial Clustering of Applications with Noise", International Conference on Service Systems and Service Management, Chengdu, pp. 1-4, 2007, doi:10.1109/ICSSSM.2007.4280175.

[12] B. Liu, "A Fast Density-Based Clustering Algorithm for Large Databases," International Conference on Machine Learning and Cybernetics, Dalian, China, pp. 996-1000, 2006, doi: 10.1109/ICMLC.2006.258531.

[13] Tsai, Cheng-Fa & Wu, Chien-Tsung. "GF-DBSCAN: A new efficient and effective data clustering technique for large databases." In Proceedings of the 9th WSEAS international conference on Multimedia systems & signal processing, pages 231–236, 2009, ISBN: 978-960-474-077-2.

[14] S. Mahran and K. Mahar. "Using grid for accelerating density-based clustering." 8th IEEE International Conference on Computer and Information Technology, Sydney, NSW, 2008, pp. 35-40, doi: 10.1109/CIT.2008.4594646.

[15] Zhou, S., Zhou, A., Jin, W., Fan, Y. ning Qian, W. "FDBSCAN: A fast DBSCAN algorithm." In: Federation, C.C. (ed.) Journal of Software, pp. 735–744, Science Press, Beijing, 2000.

[16] Roy, S., Bhattacharyya, D. K. "An approach to find embedded clusters using density based techniques. "In Distributed Computing and Internet Technology, pp. 523-535, 2005, doi:10.1007/11604655_59

[17] P. Viswanath and R. Pinkesh, "I-DBSCAN: A Fast Hybrid Density Based Clustering Method." 18th International Conference on Pattern Recognition (ICPR'06), Hong Kong, pp. 912-915, 2006, doi: 10.1109/ICPR.2006.741.

[18] Derya Birant, Alp Kut, "ST-DBSCAN: An algorithm for clustering spatial–temporal data," Data & Knowledge Engineering, Volume 60, Issue 1, Pages 208-221, ISSN 0169-023X, 2007, https://doi.org/10.1016/j.datak.2006.01.013.

[19] K. Tamura and T. Ichimura, "Density-Based Spatiotemporal Clustering Algorithm for Extracting Bursty Areas from Georeferenced Documents." IEEE International Conference on Systems, Man, and Cybernetics, Manchester, pp. 2079-2084, 2013, doi:10.1109/SMC.2013.356.

[20] Nitta N., Kumihashi Y., Kato T., Babaguchi N. "Real-World Event Detection Using Flickr Images." In: Gurrin C., Hopfgartner F., Hurst W., Johansen H., Lee H., O'Connor N. (eds) MultiMedia Modeling (MMM) Lecture Notes in Computer Science, vol 8326. Springer, Cham. 2014, doi: https://doi.org/10.1007/978-3-319-04117-9_29.

[21] POPOVICI, Robert, Andreas WEILER, Michael GROSSNIKLAUS. "On-line Clustering for Real-Time Topic Detection in

Social Media Streaming Data." SNOW 2014 Data Challenge. Seoul, Korea, Apr 8, 2014. In: PAPADOPOULOS, Symeon, ed. and others. Proceedings of the SNOW 2014 Data Challenge co-located with 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, pp. 57-63, April 8, 2014 http://ceur-ws.org/Vol-1150/popovici.pdf

[22] Markus Götz, Christian Bodenstein, and Morris Riedel. "HPDBSCAN: highly parallel DBSCAN." In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments (MLHPC '15). Association for Computing Machinery, New York, NY, USA, Article 2, 1–10, 2015, doi:https://doi.org/10.1145/2834892.2834894

[23] Sándor Szénás, "Parallel implementation of DBSCAN algorithm using multiple graphics accelerators," 16th SGEM geoconference on informatics, geoinformatics and remote sensing section informatics 28 june – 7 july, 2016, Bulgaria.

[24] W. Chen, Y. Song, H. Bai, C. Lin and E. Y. Chang, "Parallel Spectral Clustering in Distributed Systems" In IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 3, pp. 568-586, March 2011. doi: 10.1109/TPAMI.2010.88.

[25] Ling Liyang SONG Hongzhen WANG Shen Liu Jinyu, "Parallel Implementation of DBSCAN Algorithm Based on Spark," ,2019, https://www.cse.ust.hk/msbd5003/pastproj/deep1.pdf.

[26] Luo, G., Luo, X., Gooch, T.F., Tian, L., & Qin, K. "A Parallel DBSCAN Algorithm Based on Spark*,"* IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom*)*, pp. 548-553, 2016, doi: 10.1109/BDCloud-SocialCom-SustainCom.2016.85.

Authors' Profiles

Maithri. C. is an Associate Professor in Department of Computer Science and Engineering at Kalpataru Institute of Technology, Tiptur. She is currently pursuing PhD under VTU, Belagavi. She has published several research papers in International Conferences. She is an active member in ISTE.

Dr. Chandramouli H. received his PhD in the year of 2014 and currently working as a Professor in the Department of Computer Science and Engineering at East Point College of Engineering and Technology, Bengaluru. He has 22 years of rich experience in the academics. He has published more than 25 research articles in National and International Journals. He holds CSI membership and an active member in CSI events. His research area includes wireless sensor network, Resource allocation in Networking, Big Data Analytics.

How to cite this paper: Maithri. C., Chandramouli H., "Parallel DBSCAN Clustering Algorithm Using Hadoop Map-reduce Framework for Spatial Data", International Journal of Information Technology and Computer Science(IJITCS), Vol.14, No.6, pp.1-12, 2022. DOI:10.5815/ijitcs.2022.06.01