# Efficient Parallel DBSCAN algorithms for Bigdata using MapReduce

*Thesis submitted in partial fulfillment of the requirements for the award of degree of*

**Master of Engineering**
in
**Software Engineering**

*Submitted By*
**Risha Gulati**
**(Roll No. 801431021)**

Under the supervision of:
**Dr. Rinkle Rani**
Associate Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

**June 2016**

## CERTIFICATE

I hereby certify that the work which is being presented in the thesis titled, "*Efficient Parallel DBSCAN Algorithms for Bigdata Using MapReduce*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Rinkle Rani and refers other's research work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for the award of any other degree of this or any other university.

(Risha Gulati)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

14/06/16

(Dr. Rinkle Rani)
Associate Professor,
Computer Science and Engineering Department,
Thapar University, Patiala

**Countersigned by**

13/6/16

**(Dr. Deepak Garg)**
Head,
Computer Science and Engineering Department,
Thapar University,
Patiala

**(Dr. S. S. Bhatia)**
Dean (Academic Affairs),
Thapar University,
Patiala

## Acknowledgement

The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose constant guidance and encouragement secured the success.

With the profound sense of gratitude and heartiest regard, I express my sincere feelings to my guide **Dr. Rinkle Rani,** Associate Professor, Computer Science and Engineering Department, Thapar University, Patiala, who has been very concerned and has aided for all the materials essential for the preparation of this thesis report. She has helped me to explore this vast field in an organized manner and provided me with all the ideas on how to work towards a research oriented venture. She has been a source of inspiration for me.

I am also thankful to **Dr. Deepak Garg**, Head of Computer Science and Engineering Department and **Dr. Rupali Bhardwaj**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there in the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my parents, friends and the almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Risha

**Risha Gulati**

**(801431021)**

# Abstract

Over the past 20 years, there is a tremendous increase in the data generated from various diverse sources. With the advancement in newer technologies, accumulation of digital data is growing at very high pace. A series of challenges emerges due to this huge data storage and also, making the operations such as querying, retrieving and analyzing the data very difficult and tedious. Conventional database query methods and analytical technologies are becoming insufficient to deal with this huge amount of data. Cluster analysis has become the important data analysis method to unveil unknown patterns from the data. During the last few years, the mining of relational databases has become the popular research topic. Various clustering algorithms such as Partitioned, Hierarchical etc. have been proposed for this type of databases, but only a few methods are proposed for spatial databases. Spatial clustering has become an active topic for researchers in spatial data mining.

This research explores the usage of one of the clustering method i.e. Density-Based Clustering, for spatial data mining. DBSCAN is the most popular algorithm in this sub-type of clustering method. The existing algorithm has quadratic time complexity which can be further reduced by using indexing structure to O(nlogn). But this algorithm is not able to handle very large databases and takes a significant amount of time. The existing algorithms are implemented in R-tool and comparison is done between these algorithms on different datasets. Modifications have been proposed in the existing algorithm and a new algorithm is proposed named as MR-DBSCAN-KD implemented with the usage of MapReduce programming model. A traditional DBSCAN is also implemented on Hadoop framework. Both the algorithms are tested on multinode Hadoop framework by connecting three nodes. An analysis has been done between these algorithms on the basis of execution time and number of clusters performed. It has been experimentally verified that this algorithm can work efficiently with large databases and reduces the execution time.

**TABLE OF CONTENTS**

# List of Figures

# List of Tables

# Chapter 1 Introduction

## 1.1 Data Mining

### 1.1.1 Introduction to Data Mining

[2]With the advent of new technology, the amount of data generated increases in almost all the fields and so the demand to store and process this huge data is increasing. There are multiple sources of data such as data generated by the government like a recently computed census, data collected online when you buy any product from the supermarket such as Amazon, Flipkart, etc., data generated from scientific experiments, banking transactions, business domain. So, the volume of data generated from these sources is almost incomprehensible. The main problem is to extract the valuable information from this raw data.

Traditional approaches are used before for dealing with the data which are mainly statistical methods including Fourier Transforms, Multivariate Regression Analysis, etc. But due to the drastic increase in data, traditional methods are becoming impractical in almost all domains. This led to the need for new techniques and tools to fetch valuable information from this accumulated data. The field of Data Mining provides such techniques. Data Mining or Knowledge Discovery in Databases is the integration of techniques from various disciplines such as neural networks, information retrieval, pattern recognition, data visualization, etc.

Data Mining is defined as:

"The process of finding unknown, novel patterns from accumulated data and using it in decision making process."

Data Mining is an important step in KDD where "Knowledge" in KDD refers to the discovery of patterns extracted from the processed data. Fig 1.1 shows the steps of KDD process. Data mining aims to find hidden patterns from the data and finding relationships in the data.

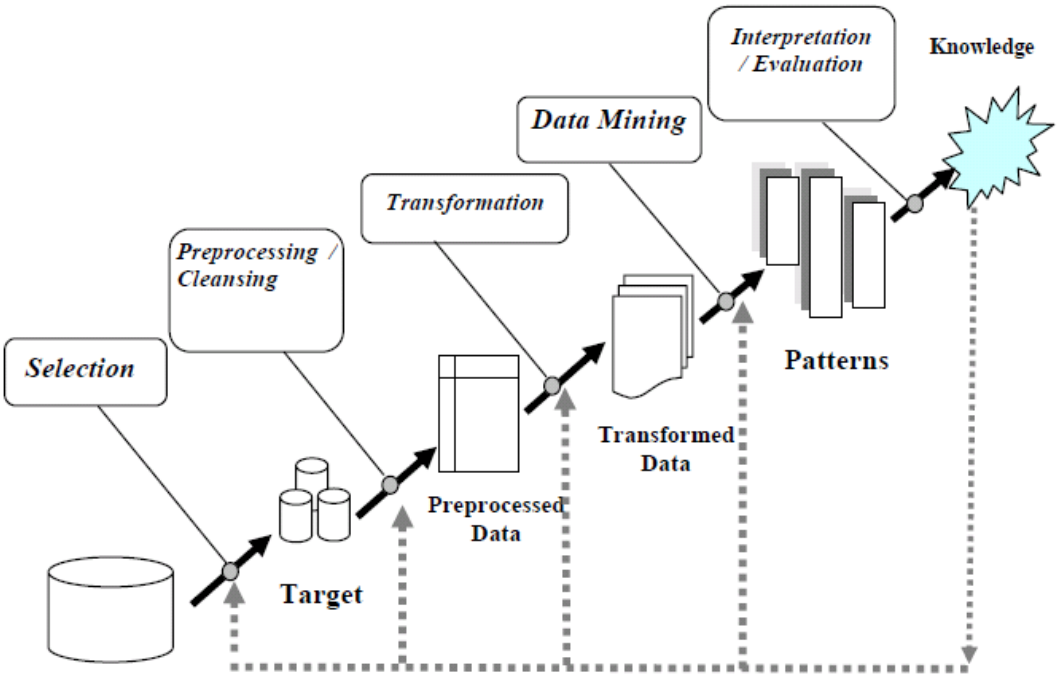


Fig.1.1: KDD Process[1]

### 1.1.2 Data Mining Process[3]

Performing data mining in any field refers to undergoing a full-fledged process which requires different approaches. On the high level, data mining process consists of mainly three components: captured data; mining and presentation of data. A consortium of users and vendors introduced a new model called CRISP-DM model which is a hierarchical process consisting of six phases.

Each phase of CRISP-DM model has some predefined tasks which are described as follows:

- Business Understanding:

This phase aims at understanding the business problem and setting the requirements for data mining project. Project objectives are converted into data mining problem and a plan for the project is prepared.

- Data Understanding

This phase focuses on gathering and analysis of initial data. Data mining experts use traditional data analysis tools to analyze the data and finding some important characteristics. They also work upon verifying the quality of data gathered.

- Data Preparation

For applying mining method, data should be in the form on which these methods can be applied. So this phase transforms the data into required format.

- Modeling

This is the phase where actual data mining exists. Various data mining methods are applied depending upon the type of data. After applying these methods, results are evaluated to find out exact parameters for data mining methods.

- Evaluation

This phase analyzes the model from the business point of view, whether results meet their expectations or not. If they are not, then they try to fit another mining model on data and again analyze it. After getting optimal values, next step is to use these results for visualization.

- Deployment

This phase works on presenting and using the results for business profits. It can produce reports specifying results or can also implement iterative DM model.
Fig 1.2 shows steps of Cross Industry Standard for Data Mining (CRISP-DM) model.

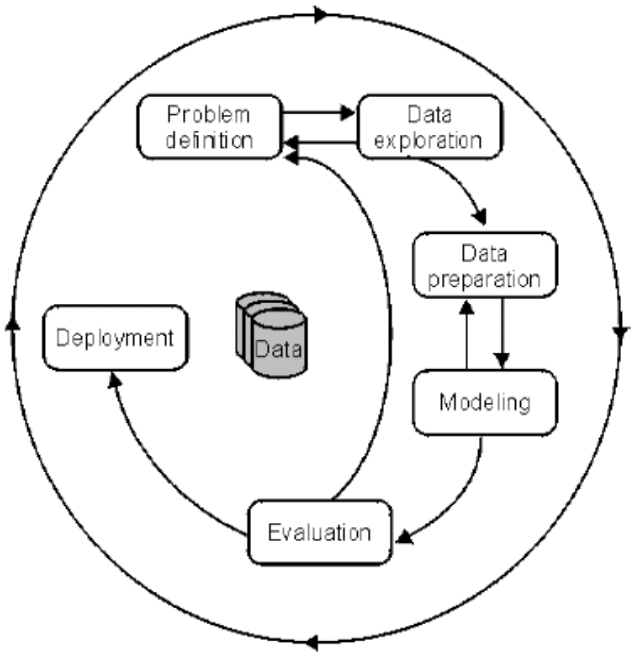


Fig.1.2: CRISP-DM Model [3]

### 1.1.3 Data Mining Techniques[3][1][4]

Data mining techniques are classified into three main categories: predictive, descriptive and link analysis.

Predictive techniques [3][4]:

These techniques predict the nature of previously unknown and hidden data using some attributes of data. Predicting the behavior of customer shopping depending upon his/her previous experiences with the particular website is a simple example for applying predictive techniques.

Following are techniques used for predictive modeling:

1. Classification[1]
2. Regression[1]
3. Deviation detection[1]

Descriptive techniques [3][4]:

These techniques focus on another application of data mining which is to describe the data in the manner understandable by a user. It also includes the task of segmenting database into different previously unknown groups which have

some similarity between them; groups having high internal similarity and low external similarity. It also presents these groups for the analysis to the user.

1. Clustering[1]
2. Summarization[1]
3. Dependency Modeling[1]

Link Analysis:[3][4]

These techniques aim to form relationships known as associations, between the records or groups of records in the database. These mainly focus on finding patterns among events depending upon some set of rules over a period of time. Market Basket Analysis is one of the important applications of these techniques and the popular algorithm called Apriori Algorithm is used for finding association rules.

1. Association Rule Discovery[3]
2. Sequential Pattern Discovery[3]
3. Similar Time Sequence Discovery[3]

## 1.2 Clustering

Clustering is a sub-type of data mining techniques. Clustering or cluster analysis aims to find groups from dataset based on some similarity measure. This measure can be a distance between the data points such that data points in a group have distance less than threshold distance i.e., these points are homogeneous. Distance between data points in different groups has more distance than the threshold distance i.e., these points are heterogeneous. These groups of data points having similar behavior are used to form clusters. Unlike classification, clustering does not require a number of clusters prior to clustering, thus, clustering is a type of unsupervised learning as it relies on "learning from observation". A most popular distance measure: Euclidean distance is usually used for clustering.

### 1.2.1 Introduction to Clustering Algorithms

Clustering Algorithms are classified into four main categories: Partitioned, Hierarchical, Grid Based and Density Based algorithms. Figure 1.3 shows the main types of clustering.

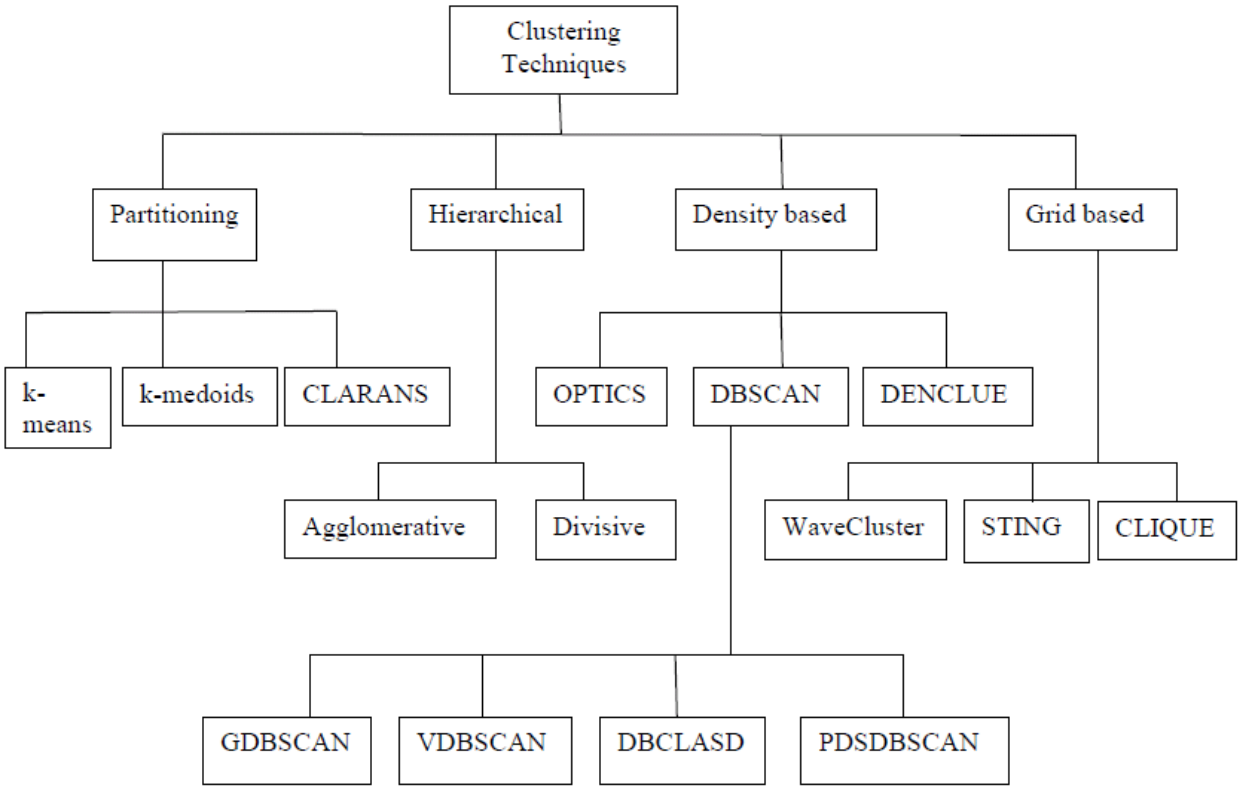


Fig.1.3: Types of Clustering

I. **Partitioning Algorithms**

Given a set of n objects, partitioning algorithms construct partition $k \leq n$, known as clusters of data where k is an input parameter. It is an iterative approach where data points are reassigned to clusters as the cluster centers are redefined. Partitioning of data is based upon some optimization criteria (i.e., minimizing the distance between data points and cluster centers) and optimizes this criterion by reallocating data points to clusters. It starts with initially k random centers and then uses an iterative approach to improve the quality of clusters by moving object from one cluster to another.

1. **K-means Method** [5][6]

   This is the most popular and widely used partitioning algorithm. Given a number K (input parameter), this algorithm partitions data into k groups

known as clusters, which are represented by K centers or centroids, ($C_1$, $C_2$, $C_3$ ……$C_K$). The centroid of each cluster is calculated as weighted average of points in a cluster. This is also known as centroid method. The criteria used are defined as follows:

$$E = \sum_{i=1}^{K} \sum_{O \in C_i} dist(p, C_i)^2$$

E, objective function, is a sum of the square of all points in the dataset and tries to minimize the distance between data points and cluster centers (centroids). Dist(x,y) used in E is Euclidean distance which is defined as:

$$D(x,y) = (\sum (x-y)^2)^{1/2}$$

K-means Clustering Algorithm:

This algorithm starts with the input parameter from user K which is the number of clusters to be formed as output and dataset on which clustering need to be performed. It outputs K clusters.

1. Select k cluster centers (centroids) randomly.
2. Find the Euclidean distance between a data point and each of centroids and assigning it to the nearest centroid to form a cluster.
3. Compute the centroids of each cluster.
4. Check if terminating condition is met or not. If cluster membership remains same, then, clustering process stops, otherwise go to step 2.

This algorithm is scalable and efficient with the time complexity of O(I * k*n) , where *i* is a number of iterations, *k* is a number of clusters and *n* is the total number of data points. This is the most important advantage of the k-means algorithm that it has linear complexity as comparison to other clustering algorithms and can also be efficient in case of very large data. In spite of this, it has some weakness that efficiency of clustering depends on input parameters "*k*" and results of algorithms are sensitive to outliers or noise. Also, it does not find arbitrary shaped clusters. Various versions of k-means algorithms are suggested to improve its weakness.

2. **K-Medoids Method** (Partitioning Around Medoids, PAM) [7][8]

   K-Medoids is similar to k-means algorithm and is just an extension of it. It overcomes drawback of k-means by handling outliers efficiently. Rather than finding centroid of clusters, it finds Medoids of clusters which is most centrally located point in the cluster and used for representing clusters. The algorithm starts with finding initially k Medoids and assigning data points to these Medoids depending upon distance between points and Medoids. After, it swaps Medoids with non-Medoids points and finds new rearrangement of clusters until desired membership is obtained. It uses Medoids as they do not depend upon extreme values whereas while calculating mean of clusters, it needs to consider that values also.

   The complexity of this algorithm is $O(Ik(n-k)^2)$ where *I* is the number of iterations of the algorithm. The algorithm becomes inefficient with increase in number of *n* and *k* due to square factor in complexity. Thus, it is efficient only for small datasets.

3. **CLARA (Clustering LARge Applications**)[4][7][9]

   CLARA was developed by Kaufman & Rousseeuw (1990) and overcomes the weakness of k-Medoids algorithm by the method of sampling. Instead of finding Medoids for whole dataset, it finds samples of datasets of size $s$ and applies PAM on each of the obtained sample by finding medoids of each of the sample. If random sampling has been performed in sufficient way, then medoids of sample can approximate the Medoids of whole dataset. So, CLARA creates multiple sample and produce best result out of this. For example, if medoids of whole dataset is not contained in medoids of samples, then it does not produce best results. So, efficiency of CLARA depends on the sample size *s*. In [KR], experiments shows that CLARA produces good results with 5 samples of *40+k* size. The complexity of each iteration of algorithms is $O(k * s^2 + k * (n-k))$ where *s* is sample size and *n* is size of dataset. As compared to complexity of PAM, *(n-k)* does have any power so producing linear time complexity.

4. **(Clustering LARge ApplicatioNS**) [7][9]

This algorithm works on the same principle as of CLARA and is based upon randomized search. Ng and Han (1994) proposed CLARANS to find clusters in dataset as a search problem in graphs where nodes of graphs represent set of *k* object indicating selected medoids. Two nodes are adjacent if they differ by only one object. Similarly, PAM can also be viewed as a problem of graph to find the minimum in it. Neighbors are examined to find the node with which it can be replaced. This process continues until minimum is obtained. CLARA also uses the same approach as of PAM but it only searches in subgraph examining fewer neighbors than PAM. Subgraphs are the samples obtained and CLARA search for the minimum in many subgraphs which is a very time-consuming process. On the other hand, CLARANS also search in subgraphs like CLARA do but, it differs from CLARA in a way that CLARANS assumes subgraph as a sample of neighbors in previous search whereas CLARA assumes subgraphs as a sample of nodes at the start of search process. In worst case, CLARANS compares a data point with every other point which makes its run-time$\mathrm{O}(kn^2)$. So, this is not suitable for large datasets due to its high complexity.

## II. Hierarchical Methods

Hierarchical clustering produces a series of nested clusters as compared to partitioning clustering which produces only flat set of clusters. It is recursive process where data points are breakdown into smaller sets in either top down manner or bottom up manner. The structure of clusters is obtained in the form of tree which is graphically represented by dendogram tree. Dendogram shows the nested grouping of data points and similarity level at which groupings change [6]. Figure1.4 shows two-dimensional data consisting of 7 data points labeled as A, B, C, D, E, F and G in three clusters. A dendogram showing clusters of figure 1.5 using single link algorithm is shown in figure 4.

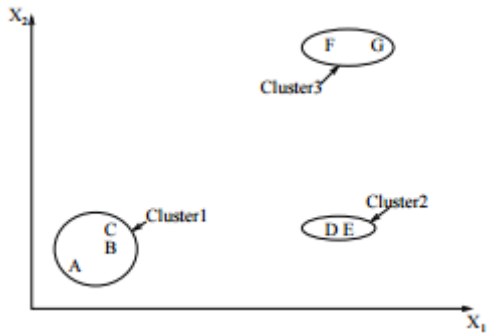


Fig 1.4: Points in three clusters [6]

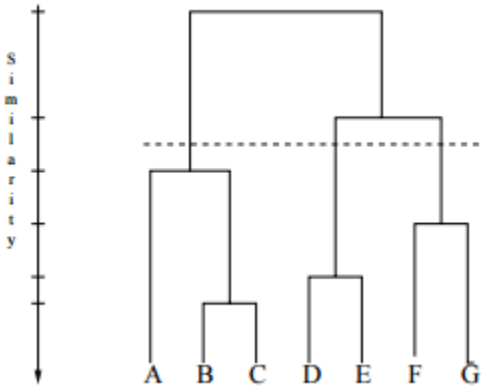


Fig 1.5: Dendogram using single-link method [6]

1. **Traditional Linkage algorithms[5]**

   The hierarchical algorithms are classified into two main categories: Agglomerative (bottom-up) approach and Divisive (top-down) approach. These both approaches find similarity measure among the data points to decide whether they can be merged into one cluster or not. For this, they find proximity matrix which computes distance between the data points. There are several ways suggested to compute this matrix which are discussed as follows:

   A. Single Linkage: This is the simplest and the most popular method used. In this, distance between two clusters is determined by finding distance between data points x and y where x belongs to first cluster and y belongs to second cluster. Similarity between clusters is determined by minimum distance between any of the two data points belonging to different clusters. It is also called as

nearest neighbor method. It suffers from chaining effect and tends to find clusters elongated in shape.

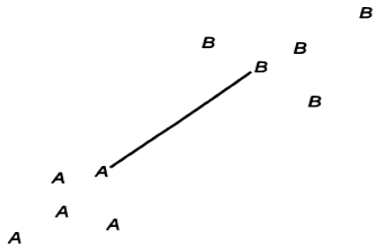


Fig 1.6: Single Link [5]

B. Complete Linkage: In this method, distance between two clusters is defined as the maximum distance between any point of first cluster and any point of second cluster. It is also known as farthest neighbor method. It tends to find clusters which are compact or tightly bound. It finds more useful hierarchy of clusters than single linkage methods.

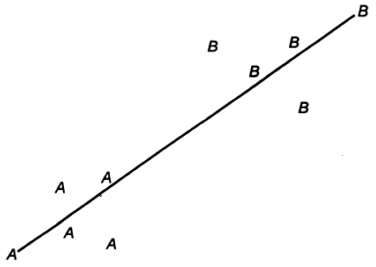


Fig 1.7: Complete Link [5]

C. Average Linkage: In this method, distance between two clusters is determined by the average of all pair wise distances between data point of first cluster and data point of second cluster. This approach works well and can also join clusters with small variances.

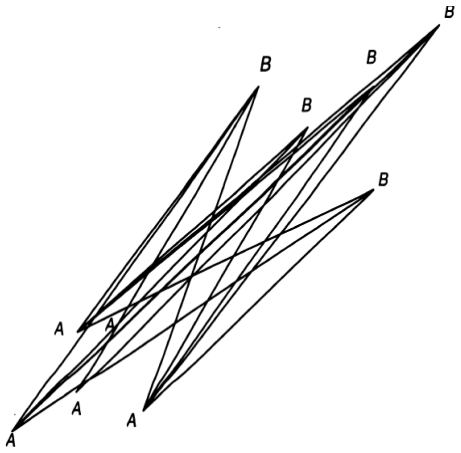


Fig 1.8: Average Link [5]

## 2. Main Approaches

These three methods can be used by agglomerative and divisive approach for computing distance between clusters. Based upon these methods, the two main approaches are discussed below [4]. Algorithms under this category are BIRCH [19], CURE [44] and Chameleon [43].

1. Agglomerative Approach[6]: It is a bottom-up approach where initially we have *n* data points each representing clusters and these clusters are merged based on some similarity measure (single link or complete link) forming larger and larger clusters. This process of merging continues until the desired number of clusters is obtained or all data points belong to one large cluster- top level of hierarchy. Following are the steps of this algorithm[5]:
    1. Place each data point in its own cluster. Thus, we have $n$ clusters initially.
    2. Construct a matrix by computing distance between pairs of all clusters using distance measure such as single link or complete link measure.
    3. Sort these distances in ascending order.
    4. Find the two clusters having smallest distance and merge them forming newer and bigger cluster.
    5. If all the data points belong to one single cluster, then, stop.
    6. Compute new distance matrix between new clusters obtained and go to step 3.

2. Divisive Approach [6]: It is top down approach and works in opposite to agglomerative. It starts with the whole dataset in one cluster and recursively splitting it into smaller clusters until all data points belong to different clusters or terminating condition is met. The most important step is to how to split clusters into smaller ones. This approach is similar to divide and conquer problem. It also uses distance matrix to split clusters. Following are the steps of this algorithm[5]:
    1. Decide a method to measure the distance between the points and set threshold distance.
    2. Compute distance matrix by finding the distance between all pair of points and sort them in ascending order.
    3. Find the most dissimilar points based upon the distance between them. Larger the distances, dissimilar are the points.
    4. If there are no dissimilar points left and there is no cluster left for splitting, then stop otherwise continues.
    5. For further splitting, k-means method is used.
    6. If there is only one point in each cluster, then stop, otherwise go to step 2.

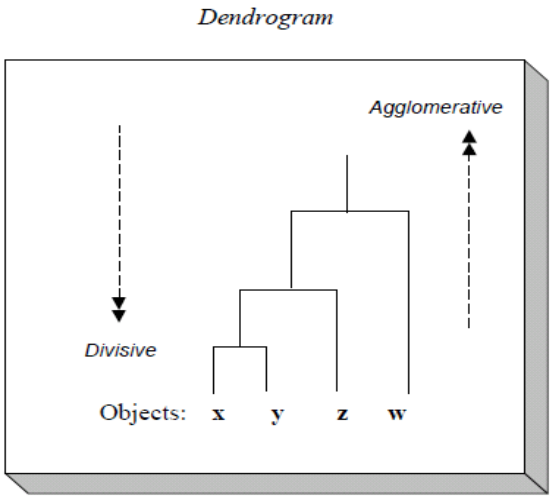


Fig 1.9: Hierarchical Clustering [7]

## III. Density- Based Methods

Density-based methods are based on the assumption that clusters are regions of high density and are separated from regions of low density. So, the aim of these methods is to find these dense areas. The basis for density-based method is that for a data point to be in a cluster, it must have a minimum number of points in the range of certain distance. Points which are not in this denser region are known as outliers or noise. These algorithms are classified into two main categories: Density-Connectivity algorithms and Density function algorithms. DBSCAN and OPTICS are the most popular algorithms under the category of density connectivity and DENCLUE comes under the category of the density function.

1. **DBSCAN (Density Based Spatial Clustering of Applications with Noise)[11]**

   This is the most popular density based algorithm used for mining spatial data. It requires two input parameters from a user which is: size of neighborhood(Eps) and minimum number of points in neighborhood(N). The algorithm starts by finding points in the neighborhood of each point in the radius of Eps. If the neighborhood of each point contains N points, then it is labeled as core point and other points are found connected by above points to form a cluster. Objects are labeled as border points if they have fewer points than N in their neighborhood. Points left are known as outliers or noise.

   DBSCAN finds clusters of arbitrary shape and able to find clusters in the high dimensional spatial database. It requires user to specify input parameters which can be a tedious task and may affect the clustering. Ester at.el. [11] uses spatial index for finding neighbors in the effective manner which improves its time complexity from $O(n^2)\ to\ O(n\log n)$.

2. **OPTICS(Ordering Points To Identify Clustering Structure)[10]**

   OPTICS proposed by Ankerst [10], is an extension of DBSCAN and works on the same approach as of DBSCAN. DBSCAN has weakness that clustering output is sensitive to input parameters so

that different input parameters provide the different number and different arrangement of clusters. OPTICS overcomes this weakness by creating an ordering of points which can automatically extract clusters in data. The time complexity of OPTICS is similar to DBSCAN *O(nlogn)* in the case of indexing structure.

### 3. DENCLUE(DENsity Based CLUestEring)[12]

DENCLUE works on a different approach from DBSCAN and OPTICS and uses density function for finding clusters in data. It assumes that objects are influenced by other objects and uses influence function to find it. Density is determined as the sum of influence function of all the objects. Cluster membership is decided with the help of density attractors which are local maxima of density functions. DENCLUE uses different influence functions, so able to generalize partitional, hierarchical and density-based clustering algorithms depending upon the choice of this function. It can handle outliers very well and can work efficiently on high-dimensional datasets.

## IV. Grid Based Methods

Grid based methods are based on space partitioning rather than data partitioning. Space partitioning is based on the grid characteristics of the input data whereas data partitioning is about data membership in regions resulted from space partitioning. These methods quantize the datasets into finite number of grids and work on data points belonging to these cells of grids. By this way, they become independent from data ordering and can work with data of different data types. Merging of cells in grid and cluster membership is decided by predefined parameter but not by any distance measure as discussed in another approaches. Traditional grid based algorithms are WaveCluster and STING.

1. **STING** [13][7]

The algorithm STING (Statistical INformation Grid-based methods) proposed by Wang, 1997[14] uses hierarchical structure to break the spatial data space into number of cells. They stores statistical information about data in nodes of trees where nodes represent grid cells. For each node in tree, it computes point count and attribute-dependent measures: mean variance, minimum, maximum and type of distribution. These measures are summed up as we go higher in the hierarchy as minimum of certain node is equal to minimum of its children. Figure 1.10 represents cell generation and formation of tree structure.

STING is highly scalable as new cells can be inserted in the grid easily and region queries can be resolved by scanning only appropriate cells at each level of hierarchy. During cluster formation, grid cells need to be merged depending upon some parameter where only parents are merged but not children and result in formation of clusters having vertical and horizontal boundaries. When Grid is constructed in $O(n)$ time, some cells are identified and connected to form clusters. It uses multi-resolution method that depends upon the number of leaves in tree.

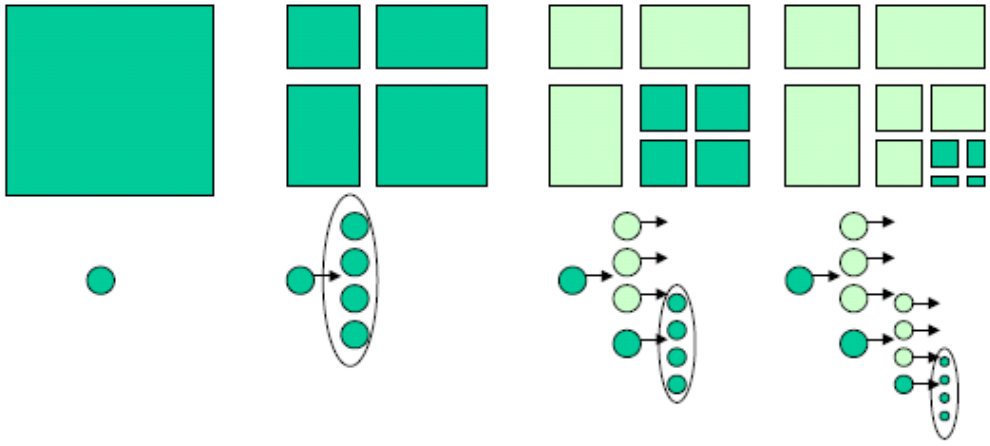

Fig 1.10: Cell generation and tree structure [13]

**2. WaveCluster[7]**

WaveCluster is clustering approach which uses different strategy and uses wavelets transforms and multi-resolution method. It uses multi-resolution approach of wavelets to find arbitrary shaped clusters at different levels of resolution. A wavelet transforms is signal processing method that uses various frequency bands. This method helps to find clusters of data points at different level of detail.

WaveCluster can form clusters of high quality and have time complexity of *O(n)* which makes it efficient than other algorithms. It can handle outliers effectively and able to perform clustering in high-dimensional data. Figure 1.11 shows the application of WaveCluster where leftmost image shows clustering at high resolution and rightmost image shows clustering at lowest resolution.

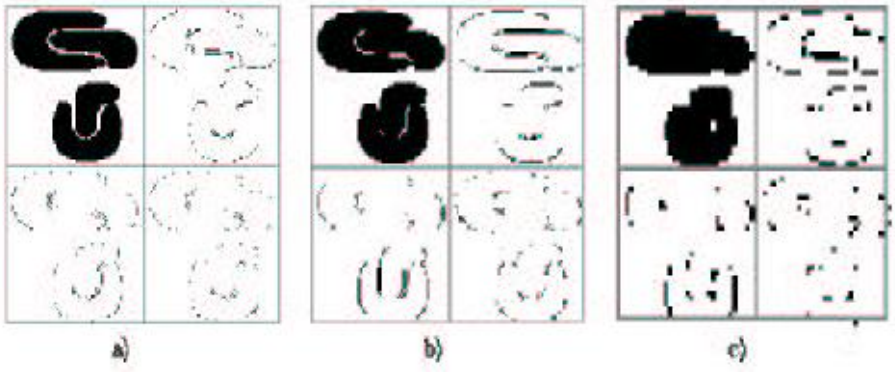


Fig 1.11: Application of WaveCluster [7]

### 1.2.2 **Clustering Algorithms for Bigdata[**15]

Clustering methods for bigdata must be chosen based on some special criteria taking into consideration three-dimensional properties of bigdata: volume, velocity and variety. Algorithms selected under this category are: BIRCH[ 19], DENCLUE[12], EM[17], OptiGrid[18], FCM[16].

**I. FCM(Fuzzy-CMeans)[16]**

This is the most popular algorithm in the area of fuzzy clustering. Fuzzy clustering is a type of soft clustering in which each object is associated with every cluster with some degree of membership. Thus, each object may belong to many clusters with some degrees of belief. It differs from

hard clustering where each object belongs to only one cluster. Its basic approach is same as of K-Means algorithm where it iteratively finds cluster centers and decides the membership of data points. It defines cluster center as the most characteristic point in a cluster and finds membership degree to clusters with respect to this center. It differs from K-Means in the sense that rather than defining hard clustering about to which cluster a point belongs to, it assigns a value ranging from 0 to 1 to each object to measure the likelihood of the object belonging to the particular cluster. Higher membership value means more likely a point will belong to that cluster. A generalized version of FCM has been proposed through a family of objective functions.

**II. BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)[19][7]**

This algorithm uses a hierarchical structure for building dendogram known as CF-trees (Clustering Feature tree). Cluster feature is the triple stored for each of the data points which consists of three values: *n* is the number of data points in the cluster, *l* is the linear sum of the attribute values of the data points in cluster and *s* is the sum of squares of attribute values of the data points in cluster. These triples are stored in a tree which is known as CF-tree. It is used for incremental and dynamically clustering for data points where it does store the whole dataset in memory but only important measures. CF-trees have two parameters which are Branching factor *B* and Threshold *T*. B is the maximum number of children of a non-leaf node and T is the maximum distance between any pair of data points in cluster stored at the leaf node. BIRCH has an important advantage that it can cluster dataset with just only one scan of dataset making its complexity of *O(n)*. It can handle noise very well but not scalable to high-dimensional datasets. Also, it is sensitive to the order of input value, so can create different clusters for same input values.

### III. DENCLUE[12]

DENCLUE is important algorithm under the category of density-based clustering methods and performs clustering for spatial data. It uses density function to assign membership to clusters through finding influence function of each data point on another and density function is summed up to the influence function of all the data points. Density attractors are labeled as clusters which are local maxima of a density function. Even though it requires two parameters from a user; it has a strong mathematical foundation with which it generalizes other clustering methods by choosing different density function. It can handle noise very well and store information in grid cells which actually contain data points. It can effectively perform clustering in very high dimensional datasets and is faster than DBSCAN due to the use of tree-based structure for storing grid cells information.

### IV. OptiGrid[18]

OptiGrid is grid based clustering approach where it focuses on space partitioning. It obtains optimal grid partitioning through a set of selected projections and cutting plane obtained through these projections. After the construction of grid, it finds clusters from the dataset using density function. The algorithm is applied to these clusters in a recursive manner and in each recursive step; it only maintains dense regions of the grid. It can cluster datasets of high dimensions but fail to perform, if high dimension have clusters embedded in low dimension space. Also, OptiGrid is sensitive to initial selection of parameters and projections and finds hard to find optimal grid partitioning.

### V. EM (Error Maximization)[17]

This algorithm comes under the category of Model-based clustering and is the extension of K-means algorithm. EM decides the membership of clusters using probabilistic distribution function and finds the best suitable parameter for this function in an iterative manner. It starts with randomly labeling data point’s clusters and then estimates initial parameters from

this labeled data. Then, it iteratively estimated the unknown parameters in two steps: E-step, the Expectation step and M-step, the Maximization step. It finds local maximum of the estimated parameters. For the correct estimate of parameters, number of iterations need to be done which is time-consuming making this algorithm expensive. Also, an algorithm is sensitive to the initial selection of parameters and can produce less realistic results in finite steps.

## 1.3 Tools/Platforms/ Languages used for Bigdata

### 1.3.1 Hadoop and MapReduce

#### I. HDFS[25]

Hadoop cluster uses Hadoop Distributed File System for storage and management of data. Its main features are: high fault tolerant, high throughput, large capacity to store very large datasets and streaming access to stored data. It does not require special hardware requirements but can be built with commodity hardware. All operations such as read/write are performed in blocks whose default size is 64MB. It follows master/slave architecture where master is known as namenode and slaves are known as datanodes. Figure 1.12 shows hadoop architecture showing its components.

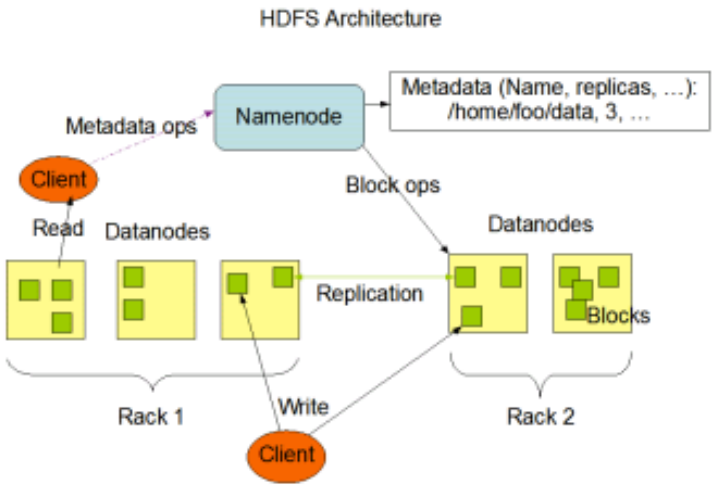


Fig 1.12: Hadoop structure Source: http://hadoop.apache.org

A namenode is the master node which manages and maintains the blocks present on datanode. It does not store blocks of data itself but maintains mapping process for locating files to datanodes where they are located. It manages all namespace operations such as opening and closing of files and directories. For reading data from datanode, client first contact namenode for location of data blocks and then read data from that datanode. For writing data, clients contact namenode and then write data to datanode in pipeline manner. Thus, it is the centralized service operating which serve client's requests. It requires very reliable and expensive hardware as it is a single point of failure. It stores namespace information such as location of files in datanodes, known as metadata in RAM. There also exists secondary namenode which periodically read meta data from RAM of namenode and writes data hard disk. It is not a substiuent of namenode but only stores metadata.

Datanodes, known as slaves, can be in large number in a cluster whereas there is a single namenode for each cluster. These are deployed on each machine in a cluster and provide the actual storage of HDFS blocks. During operation, datanodes sent heartbeats to namenode to send its status that it is online. If namenode does not receive heartbeat from any of the namenode, it assumes that datanode is not working. In this case, data stored at this datanode is lost and namenode replicate the lost data to another datanode. The information carried in heartbeats comprises of total storage capacity of datanode, capacity in use and number of transfers in progress. This information will be used by namenode for assigning data blocks to datanodes and in load balancing decisions.

### II. MapReduce[26]

MapReduce is programming model built for computing large datasets. There are two versions of MapReduce available: MapReduce 1.0 and MapReduce 2.0 (Yarn). MapReduce comprises of two components: map function and reduce function. MapReduce job splits the huge data into independent chunks which are fed to map function running in a parallel

manner. The framework itself sorts the output of map function and fed the intermediate data to reduce function. In map function, the input is fed in the form of key-value pair for computation and it outputs intermediate key-value pair. Reduce function takes this intermediate data and process these key-value pairs to produce new key-value pairs by merging all values corresponding to same key. The output from reducers is actual output of MapReduce process. There exists shuffle phase which performs the task of sorting and grouping of output data from map function. Figure 1.13 shows all three phases.

$$Map: (k1, v1) \rightarrow (k2, v2)$$

$$Reduce: (k2, v2) \rightarrow (k3, v3)$$

MapReduce consists of centralized JobTracker which performs the task of splitting input data, scheduling each job on a cluster node, monitoring jobs and re-running the failed tasks. On each cluster node, TaskTracker runs which are known as slaves and executes the task as directed by master JobTracker. TaskTracker periodically contacts JobTracker to give the status in process and to request new tasks.

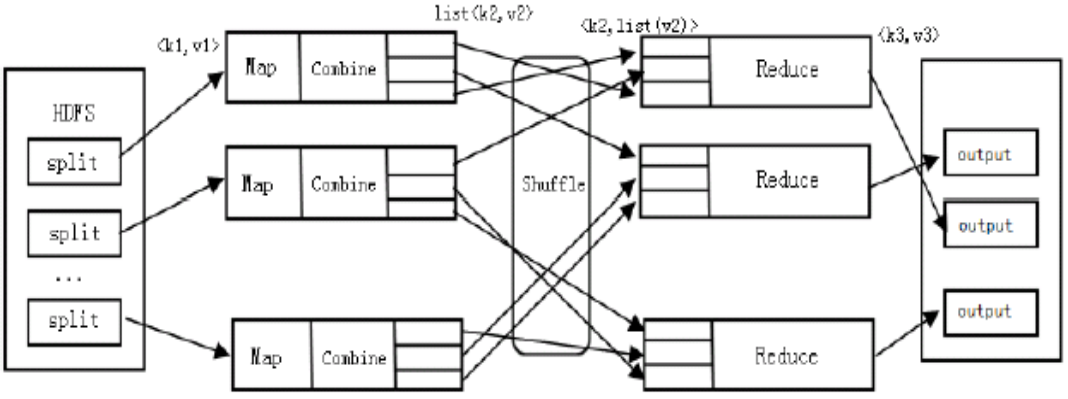


Fig 1.13: MapReduce model[35]

### 1.3.2 R and Its Packages

R is a framework for statistical analysis and graphics. It is environment as well as programming model created by Ross Ihaka and Robert Gentleman. It is a GNU project which is an implementation of S language created by John Chambers at Bell Laboratories. R and its libraries provide a wide variety of statistical as well as graphical functions. The main feature of R is that it is extensible with the help of "packages". There are lots of packages provided with the R distribution and much more can be downloaded from the internet site of CRAN (Comprehensive R Archive Network). Different packages used for implementation of data mining algorithms are as follows:

- fpc- fpc[21] is a package which implements various clustering algorithms. It also comprises of methods for cluster validation and for estimating a number of clusters. It implements a simplified version of DBSCAN without using indexing structure.
- Dbscan[20]- it consists of reimplementation of DBSCAN algorithms which is fast as compared to DBSCAN implemented by the fpc package. Dbscan implements several density based methods for spatial data including OPTICS clustering algorithms and LOF (Local Outlier Factor). Algorithms use indexing structure such as kd trees for the faster query of neighbor search.
- microbenchmark[22]- It provides methods to accurately measure the execution time of various R expressions in different units. It also provides methods to graphically view execution time of different expressions at the same time for making the comparison between them.
- cluster[23]- cluster package implements methods for cluster analysis. It provides an implementation for all of the clustering methods such as hierarchical, partitional. It also provides some datasets which can be used for clustering process.

- ggplot2[24]- It is the graphical module for R. It implements grammar of graphics and is known as plotting system for R. It inherits good parts of base and lattice graphics and plot can be built from multiple sources in steps.

## 1.4 Structure of The Thesis

The rest of the thesis is organized in the following order:

- **Chapter 2 Literature Review -** It gives the review of the three main algorithms suggested under the category of Density-Based clustering algorithms. It also discusses other variants of these main algorithms and provides the comparison of the three main approaches.
- **Chapter 3 Research Problem –** This section of thesis tells about the need for designing new algorithm. It discusses the circumstances which led to formulate the problem and set the objectives to achieve problem stated. It also specifies the methodology needed to solve the research problem.
- **Chapter 4 Design of Efficient Parallel DBSCAN algorithm –** In this chapter, first existing implementation of algorithms is discussed and implemented in the suitable environment and then, the design of proposed algorithm is explained in detail.
- **Chapter 5 Experiments and Results -** This chapter includes the experimental setup required for implementation, the experiments performed on different tools/technologies and comparison of the results obtained.
- **Chapter 6 Conclusion and Future Scope -** This section gives the conclusion of the thesis and summary of the contributions and the future scope.

Finally, references and list of publications have been appended.

## 2.1 DBSCAN

Density Based Spatial Clustering of Applications with Noise, DBSCAN [11] was the first density based spatial clustering algorithm. Spatial clustering is the clustering of spatial data emerged from various sources. Due to the high complexity of spatial data, the task of clustering becomes difficult in spatial databases. DBSCAN algorithm can find clusters in spatial data of arbitrary shape and can also handle noise in an effective manner. Figure 2.1 shows the shape of clusters identified by DBSCAN. In this, denser regions are classified as clusters and regions with low density are classified as noise. The notion of cluster membership is based on the concept of density connectivity. DBSCAN requires two global input parameters from user: radius (Eps) and threshold of points (MinPts).

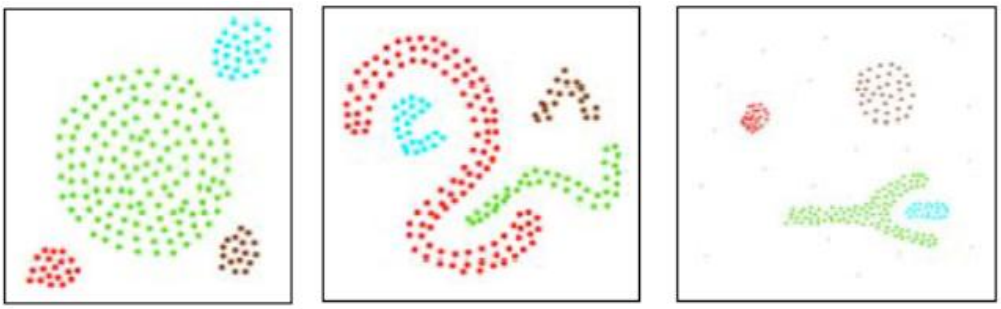

Fig. 2.1 Clustering by DBSCAN algorithm [27]

Following concepts need to be understood before moving toward details of algorithm:

Definition 1: **Eps neighborhood of point x**: are the points which lie in the radius (Eps) of point x. It gives the set of points which have distance from x in the range of Eps. It is formulated as follows:

$N_{Eps}$ (x) = y ∈ $D \mid dist(x,y) \leq Eps$

*where D is a database.*

Definition 2: **Core Point**: are the points found inside the clusters. They contain at least MinPts in the radius of Eps.

Definition 3: **Border Point**: are the point found on the border of clusters. They contain fewer points in its Eps-neighborhood than the Eps-neighborhood of core point. Different core points may share same border point.

Definition 4: **Noise Point:** are the point which does not come under the category of core and border points. They are outliers which are mostly discarded.

Definition 5: **Directly Density Reachable:** A point y is directly density-reachable from point x with respect to radius, MinPts if:

i. $y \in N_{Eps}(x)$

ii. $N_{Eps}(x) \geq MinPts$

This property is symmetric for the pair of core points.

Definition 6: **Density reachable:** A point y is density-reachable from point x with respect to radius, MinPts if there exists a sequence of point $y_1, y_2, y_3 \ldots\ldots y_n$ with $y_1 = y$ and $y_n = x$ such that $y_{i+1}$ is directly reachable from $y_i$ .

This property is an extension of direct density-reachability. This relation has transitive property.

Definition 7: **Density Connectivity:** A point y is density connected to point x if these both points are directly reachable from core point z with respect to radius, MinPts.

Definition 8: **Cluster:** A cluster *C* is subgroup of database D if it satisfies following conditions:

i. $\forall_{x,y}$ : if x belongs to *C* and point y is density-reachable from point x with respect to radius, Minpts, then, y also belongs to *C*.(Maximality Condition)

ii. $\forall_{x,y}$ : y is density connected to x with respect to radius and MinPts. (Connectivity Condition)

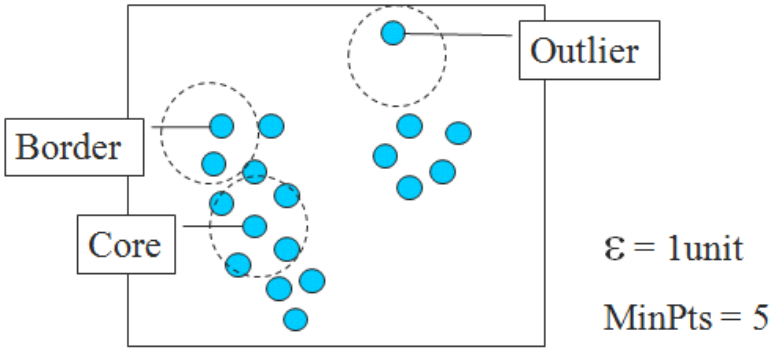


Fig 2.2: Core Points, Border Points and Outliers

**DBSCAN Algorithm[11]**: Initially, the input parameters Eps and MinPts must be known. DBSCAN starts with any random point x and retrieves all points which are density-reachable from that point. If x satisfies the condition of core point, then it forms a cluster. If x is border point, then it applies the same process on points extracted. This process continues until all the points in the database are processed at least once. Algorithm 2.1 defines the DBSCAN algorithm.

**Input**: Eps, MinPts, Input file consisting of data points
**Output**: Cluster of points

**DBSCAN** (Data File,Eps,MinPts)

1. For each of the points in dataset.
    a. If point"p" is not classified, then, call function ExpandCluster.
    b. If called function returns true, then, find another cluster.
2. **ExpandCluster: Input:** Data file, point"p", Eps, MinPts
3. Find the neighbors of point "p" in the range of Eps; call function RegionQuery(point, Eps) and store in variable "Seeds".
4. If seeds < Minpts (no core point condition is met)
    **a.** Points are added to noise.
5. Else (point"p" is core point)
    **a.** Set cluster id of all the points.
    **b.** While seeds != empty
        **i.** Fetch first point "q" from seeds.
        **ii.** Find its neighbors by calling function RegionQuery and stores in "result".
        **iii.** If result.size > MinPts
            **1.** For each of the point in result size
            **2.** If point is not classified, then, added to previous cluster id.

Algorithm 2.1 DBSCAN algorithm

**Complexity:** Steps to extract neighborhood of point called as RegionQuery in the algorithm, is the most important step which decides the complexity of algorithm. It runs this query for each of the points and takes $O(n^2)$ time. If indexing structure such as kd trees or $R^*$ trees are used, then complexity become $O(n \log n)$

**Advantages: A**dvantages of DBSCAN are as follows:

- DBSCAN does not require number of clusters to be known prior to clustering as in case of K-Means algorithm
- It can find clusters of arbitrary shape, i.e., spherical, elongated
- It can also work well with large spatial databases and can handle noise effectively

**Disadvantages:** Besides various advantages, DBSCAN have some weaknesses also.

- DBSCAN cannot handle data with varying density
- It can only produce flat structures
- Quality of clusters depends on parameters which are hard to determine.

## 2.2OPTICS

Ordering Points To Identify Clustering Structure, OPTICS [10] is an extension of DBSCAN and is the second most popular after DBSCAN. The underlying approach of clustering data is same for OPTICS and DBSCAN but OPTICS addresses some of the weakness of DBSCAN, the problem of detecting useful clusters in data of varying density and production of only flat structures. The newer algorithm computes the ordering of consisting of two values: core distance and reachability distance for each of the point in the database. Points of database are processed in some linear order such that points which are closest become neighbors in this ordering. Cluster ordering can be visualized using reachability plots which are 2D-plot with points on the x-axis arranged in order they were processed and reachability distance on the y-axis. Valleys in reachability plot indicate clusters showing points with smaller reachability value are closer. Figure 2.3 shows reachability for 2-dimensional dataset.

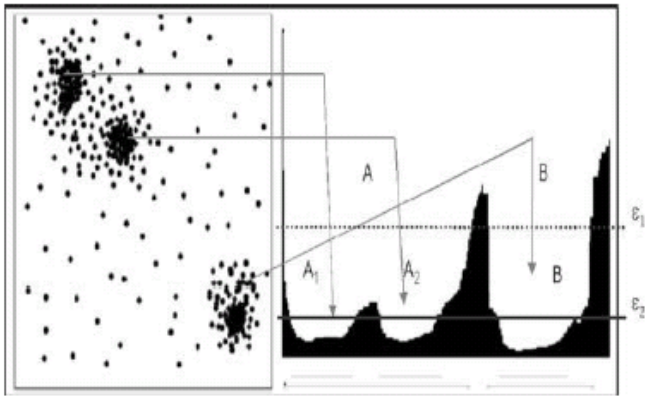


Fig.2.3: Reachability Plot for 2D dataset [28]

Definition 1: **Generating Distance:** $\varepsilon$ is the largest distance which can be considered for clusters. Clusters can be extracted for all $\varepsilon_i$ such that $\varepsilon_i$ lies in the range of 0 and$\varepsilon$.

Definition 2: **Core Distance:** Let x be any point from database D and Eps and MinPts be two input parameters. $N_{Eps}(x)$denotes the Eps-neighborhood of point x and MinPts-dist(x) denotes the distance between x and MinPts's neighbor. Core distance of point x is defined as follows:

$$\begin{Bmatrix} undefined, & Count\left(N_{Eps}(x)\right) \leq MinPts \\ MinPts - dist(x), & otherwise \end{Bmatrix}$$

It is denoted by $\varepsilon'$ and is the smallest distance which makes a point x, core point.

Definition 3: **Reachability Distance**: of point y with respect to core point x is the smallest distance such that y is density-reachable from point x. It is defined as follows:

$$\begin{Bmatrix} undefined, & \left(N_{Eps}(x)\right) \leq MinPts \\ \max\left(core - dist(x), dist(x,y)\right), & otherwise \end{Bmatrix}$$

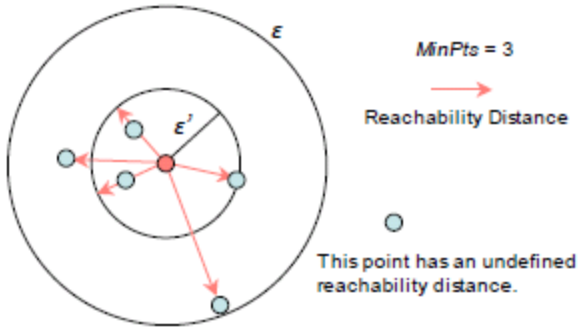


Fig. 2.4: Core distance and Reachability distance [10]

**OPTICS Algorithm[10]:** OPTICS creates an ordering of data points and storing the core and reachability distances for each of the point. This information is used by DBSCAN to extract clusters with respect to distance $\varepsilon'$ which becomes core distance and is smaller than generating distance. It uses additional data structure seedlist, which stores data points and their reachability distances arranged in ascending order.

**Complexity:** OPTICS also takes same execution time as of DBSCAN. But OPTICS requires additional data structure: Priority Queue to store core and reachability distances for each of the data point.

**Input:** Data file, Eps, Minpts

**Output:** Ordered File

**OPTICS** (Datafile, Eps, Minpts)

1. For each of the point "p" in datafile
2. If p is not processed
    a. Call function Expand Cluster
3. **ExpandCluster: Input:** set of points, point "p", Eps, Minpts
4. Find the neighbors of point "p" and store in "neighbors".
5. Find the core distance of point "p".
6. Write the point"p" in ordered file.
7. if core distance of point != undefined
    **a.** Call "Update" function to update the neighbors of point "p" and store in "seeds" file.
    **b.** While (seeds != < >)
        **i.** Fetch first element of seeds and set it to "CurrentObject"
        **ii.** Find the neighbors of CurrentObject.
        **iii.** Set core distance of CurrentObject and write in ordered file.
        **iv.** If core distance of CurrentObject is undefined
            **1.** Call "Update" function.
8. Exit.

Algorithm 2.2 OPTICS algorithm

**Update Function**: it performs the task of inserting new objects in ordered list in ascending order of their reachability distance.

**Input**: neighbors, CurrentObject

1. Get the core distance of CurrentObject in variable "dist"
2. For object from 1 to neighbors
    a. If object is not processed
        i. Find new reachability distance "rdist":max(dist, CurrentObject.dist(object)
            1. If (object.reachabilitydist == undefined)
                a. Object.reachabilitydist= rdist
                b. Insert object in ordered file
            2. Else
                a. If rdist<object.reachabilitydist
                    i. Object.reachabilitydist=rdist
3. End

Algorithm 2.3: update function of OPTICS algorithm

**Advantages:** OPTICS has following advantages:

- Ordering structure can be used to extract basic clustering information
- OPTICS does not require parameter setting as in DBSCAN
- From ordering structure, cluster membership can be decided.

**Disadvantages:** OPTICS has some weaknesses also.

- OPTICS requires a wide range of parameter setting
- It is not suitable for very high dimensional datasets
- Run-time of OPTICS is 1.6 times of DBSCAN.

## 2.3 DENCLUE

DENsity based CLUstEring, DENCLUE [12] is the algorithm designed specifically for multimedia data containing large amount of noise with arbitrary levels and is of very high dimensions. DENCLUE can be seen as generalization of other clustering methods such as partitioning, hierarchical and locality based methods. Depending upon the parameter setting, DENCLUE can provide same results as of other clustering approaches which makes it a very effective algorithm. The underlying approach of DENCLUE is different from other density based methods. DENCLUE uses two main concepts: influence function and density function. Cluster membership is based upon the closeness to denser area. DENCLUE defines two types of clusters: Centered defined clusters and Arbitrary shaped clusters. Following are the concepts which are used in this algorithm:

Definition 1: **Influence Function:** Every data point has some impact over the other data point which is modeled by influence function. Influence function of data point p $\in D$ is a function $f_B^p : D \rightarrow R_0^+$ defined in the form of basic influence function $f_B^p$

$$f_B^p(q) = f_B(p,q)$$

Some basic influence functions used are:

1. Square Wave Influence Function

$$\begin{cases} f_{square}(p,q) = 0, & if\ d(p,q) > \sigma \\ 1, & otherwise \end{cases}$$

2. Gaussian Influence Function

$$f_{guass}(p,q) = e^{\frac{-d(p,q)^2}{2\sigma^2}}$$

Definition 2: **Density Function** is defined as the addition of influence function of all data points.

Density Function resulting from Gaussian influence function is as follows:

$$f_{gauss}^D(p,q) = \sum_{i=1}^{N} e^{\frac{-d(p,q)^2}{2\sigma^2}}$$

Definition 3: **Gradient:** The gradient of density function is defined as:

$$\nabla f_{gauss}^{D}(p,q) = \sum_{i=1}^{N}(p_i - p) * e^{\frac{-d(p,q)^2}{2\sigma^2}}$$

Definition 4: **Density-Attractors:** A point $p^* \in D$, is known as density attractor, iff $p^*$ is local maximum of density function.

Definition 5: **Density Attracted Points:** A point p $\in D$ is density attracted to point $p^*$, iff there exists a path for which gradient is always positive.

For finding density attractor for point, hill-climbing procedure is used which is guided by gradient.

Definition 6: **Center-Defined Cluster:** A cluster C for the density attractor $p^*$, is subset of D if it satisfies following condition:

i. $\forall_p$ , p is density attracted to $p^*$, then, p$\in C$.

ii. $f^D(p^*) \geq \xi$.

Definition 7: **Arbitrary Shape Cluster:** these clusters are formed by merging center defined clusters if there exist a path for which density function is exceeding $\xi$.

**Algorithm:** DENCLUE also requires two input parameters on which quality of clusters depends. First parameter $\sigma$ determines the influence of point in its surrounding and second parameter $\xi$ determines the significance of density attractor. DENCLUE generalizes other clustering methods by varying the value of these parameters. In density based methods, DBSCAN can be implemented using DENCLUE with setting parameter $\sigma = Eps$ and $\xi = MinPts$. In partitioning based clustering, K-Means clustering can be implemented using Gaussian influence function and with parameter $\xi = 0$ and $\sigma = K$. In hierarchical clustering, center defined hierarchy can be obtained for the different value of $\sigma$.

DENCLUE uses only point closest to a certain point while finding density as only closest points can influence other points. All other farthest points can be neglected as their influence may be negligible. It introduces the concept of local density function based upon only nearer points. It works in two steps as discussed as follows:

1) Step1: Pre-clustering
    - Construct a map whose each dimension is 2 $\sigma$. Figure 2.5 shows map of data in 2-D space
    - Find the hypercube actually containing data points. These are known as populated cubes, $C_p$.
    - Mean of these cubes are determined and two cubes are connected if they satisfy the condition: $d\big(mean(C_1), mean(C_2)\big) \leq 4\sigma$. This step takes $O(C_p^2)$ time
    - To reduce this time, highly populated cubes are considered which satisfy the condition: $C_{sp} = c \in C_p | N_c \geq \xi_c$. As $C_p \gg C_{sp}$, time needed to connect neighbors will be: $O(C_{sp}. C_p)$.
2) Step 2: Actual Clustering
    - For clustering, only highly populated and connected cubes will be considered. This is represented by $C_r$ such that

      $C_r = C_{sp} \cup (c \in C_p | \exists c_s \in C_{sp})$ and $\exists connection(c_s, c)$.
    - Determine density attractors for each point in $C_r$, hill climbing procedure is used
    - Find the local density function with setting the limit of $4\sigma$ for finding nearer points
    - Cluster membership is defined by attaching points in the cluster within the range of $\sigma/2$ of path to the cluster
    - Points which are density attracted are directly assigned to cluster.

**Complexity:** Complexity of DENCLUE is calculated in following steps:

- To construct a grid, the full scan of the dataset is required which takes $O(D)$ time where D is a database
- To get populated and highly populated cubes, it takes $O(D + C_p.\log C_p)$ where log $C_p$ is time taken to store keys in $B^+ tree$.
- To connect neighboring cubes, it takes $O(C_{sp} * C_p)$ with $C_{sp} \ll C_p \ll D$.
- To form clusters with $C_r$ cubes, it takes$O(D_r * \log C_r)$ where $D_r$ is subset of D without outliers
- So, in average case complexity comes out to be$O(\log D)$.

**Advantages:** DENCLUE has following advantages:

- Useful for datasets with a high amount of noise
- Faster than DBSCAN by the factor of 45
- Able to generalize other clustering methods

**Disadvantages:** DENCLUE has following disadvantages:

- Hill climbing procedure takes unnecessary steps for finding density attractors
- Parameters selection is tricky task

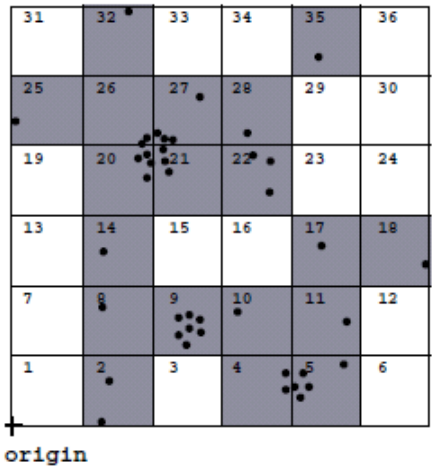


Fig 2.5 Map in 2-d space [12]

## 2.4Variants of DBSCAN

DBSCAN has various weaknesses and cannot be used in all types of datasets. Also, it requires two parameters which are hard to find. Therefore, to address these weaknesses of DBSCAN, some variation in the basic algorithm of DBSCAN are proposed. Following are some variants of DBSCAN.

### 2.4.1 DBCLASD[29]

Distribution Based Clustering of Large Spatial Data, DBCLASD is a new algorithm which addresses the problem of DBSCAN of two input parameters. This algorithm does not require any input parameters and can discover clusters of arbitrary shape. It uses an incremental approach for assigning points to the cluster. To assign a point to cluster, it only focuses on points processed so far without considering the whole dataset. DBCLASD algorithm works in two steps: generating candidates and testing candidates.

In generating candidates, points are considered which are not assigned to any cluster on the basis of region query. Points intersecting the circle of the particular radius are known as candidates. In testing candidates, the chi-square test is used. Points which lies in the range of threshold value fits the expected distribution and are right candidates. The efficiency of DBCLASD lies in between DBSCAN and CLARANS.

### 2.4.2 GDBSCAN[30]

Generalized Density Based Spatial Clustering of Applications with Noise, GDBSCAN is an algorithm which generalizes the popular algorithm DBSCAN. It has some extended features such as it can cluster points as well as polygon object using their spatial and non-spatial attributes. Spatial attributes represent the location of points or polygon in 2-dimensional space and non-spatial attributes represent additional properties such as unemployment rate of the community represented by the spatial object.

GDBSCAN generalized DBSCAN in the following manner. First, the notion of neighborhood other than distance measure can also be considered such as for finding clusters in polygon objects, intersect predicate can be used in place of the neighborhood. Second, for the second parameter of DBSCAN, MinPts, other non-spatial attributes can also be considered such as annual income of city represented by polygon. GDBSCAN can be applied to high-dimensional datasets and has application in astronomy (2D points), biology (3D points), earth science (5D points) and geography (2D polygons).

### 2.4.3 VDBSCAN[31]

Varied Density Based Spatial Clustering of Application with Noise, VDBSCAN addresses the important weakness of DBSCAN such that DBSCAN cannot discover clusters in data having varying density. VDBSCAN is introduced for the purpose of varied-density database analysis. This algorithm also selects the value of input parameter Eps automatically. The basic idea of clustering remains same as of DBSCAN but an additional step is introduced to handle variable density of data.

VDBSCAN uses k-dist plot to find the value of Eps for the different level of density. It performs clustering in two steps: finding parameters and varied density clustering. For finding parameters, k-dist are computed for each data point and plotted in sorted order. A sharp edge in the k-dist plot corresponds to the suitable value of Eps. Figure 16 shows k-dist plot to find the value of Eps and level of density. Thus, for different levels of density, a different value of Eps is determined. For each value of Eps, DBSCAN is executed to find clusters in data. It has same time complexity as of DBSCAN.

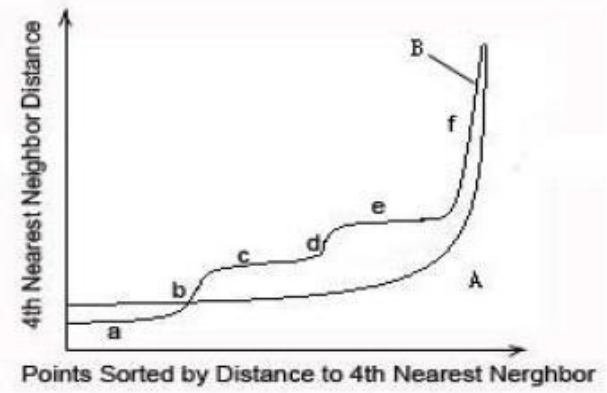


Fig. 2.6: k-dist plot to find Eps and levels of density [31]

### 2.4.4 Other Approaches

Besides above discussed algorithms, there also exist some other extensions and improvements of these algorithms. DBSCAN also have some other variants as discussed in [38], [39]. There are also approaches suggested on the basis of OPTICS algorithm as given in [40].

Popat [41] discussed RDBC, which is an extension of DBSCAN. This algorithm is based on the DBSCAN but provide better results than the DBSCAN. It calls DBSCAN with different values of Eps and MinPts and returns the clusters when their number becomes appropriate.

Xin Wang and Howard J.H. [42] have done a comparative study of two density based methods: DBSCAN and DBRS. They studied that DBSCAN does not form clusters well if the data contain non-spatial attributes, but work efficiently on other datasets. DBRS focuses on minimizing the execution time for data having variable density and can work well with non-spatial attributes. But it needs some improvement as it may fail to join certain clusters.

## 2.5 Comparison Table

This section provides the comparison among three main algorithms under density-based clustering methods. Comparison is made considering 12 different parameters such as type of data, complexity, data structure used, implementation strategy, etc. Table 1 shows the comparison between DBSCAN, OPTICS and DENCLUE.

Table 1: Comparison of DBSCAN, OPTICS, DENCLUE

| | **DBSCAN** | **OPTICS** | **DENCLUE** |
|---|---|---|---|
| **Clustering Algorithm** / **Characteristics** | Density Based Spatial Clustering Of Application With Noise | Ordering Points To Identify Clustering Structure | Density Based Clustering |
| **TYPE OF DATA** | Spatial data with noise(Numerical) | Spatial data with noise randomly distributed (Numerical) | Large number of data with large amount of noise |
| **ATTRIBUTES OF DATA** | Only spatial attributes | Only spatial attributes | Only spatial attributes |
| **INPUT PARAMETERS** | 2 input parameter- *Eps* and *Minpts* | Density Threshold | 2 input parameters- *Density Attractor* and *Influence function* |
| **VARIED DENSITY** | NO | NO | YES |
| **SHAPE OF CLUSTER** | Arbitrary | Arbitrary | Arbitrary |
| **COMPLEXITY** | O($n^2$)<br>O($n \log n$)-with index structure | O($n^2$)<br>O($n \log n$)-with index structure | O(log\|D\|) |
| **HANDLING OF NOISE** | Not very well | Similar to DBSCAN | Very Well |
| | | | |

| | | | |
|---|---|---|---|
| **DATA STRUCTURE** | R* tree | R* tree & Seed List | R* tree |
| **PURPOSE OF ALGORITHM** | To discover cluster with arbitrary shape | Generalizes clustering by creating ordering of points | Can discover other clustering algorithms like hierarchical clustering, partition based clustering etc |
| **IMPLEMENTATION STRATEGY** | Center Based Approach- density is defined for a particular point in the dataset by counting the number of points within a radius of that point which should be greater than Minpts. | Creates an ordering of the database representing its density-based clustering structure | Clusters can be discovered by determining density-attractors and uses a mathematical equation for defining density function**.** |
| **ADVANTAGES** | 1. Does not require number of clusters to be known before clustering<br>2. Works well with large spatial databases.<br>3. Handle noise effectively.<br>4. Find clusters of arbitrary shape | 1. No parameter setting as in DBSCAN.<br>2. Ordering provides basic clustering information.<br>3. Cluster assignment can be done using ordering. | 1. Generalizes other clustering algorithm.<br>2. Uses mathematical equations to define density.<br>3. Faster than DBSCAN by factor of 45. |

| | | | |
|---|---|---|---|
| **DISADVANTAGES** | 1. Produces only flat clusters.<br>2. Not able to cluster data containing different levels of density.<br>3. Quality of clusters depends on input parameters. | 1. Runtime is 1.6 times of DBSCAN.<br>2. Requires additional data structure to store ordering of points. | Hill climbing procedure uses unnecessary steps. |
| **APPLICATIONS** | Satellite images, X-Ray crystallography, Anomaly Detection in temperature data | Similar to DBSCAN | Multimedia Data:- Images ,CAD, Geographic, Molecular biology |

# Chapter 3 Problem Statement

## 3.1 Research Gaps

Over the past 20 years, there is a tremendous increase in the data generated from various diverse sources. With the advancement in newer technologies, accumulation of digital data is growing at very high pace. Due to the development of internet technologies such as high capacity servers, there is a huge amount of information and data collected every day in the fields of government, business, market, and environment. According to the report by industry experts, the volume of information will expand from 3.2 ZB to 40 ZB by 2020[Source: Top 10 amazing facts to know about big data, Bhavna Singh, Oct, 2015]. Services such as social networking, cloud storages and etc., yield a huge amount of data and implies an urgent need to store and analyze this data. A series of challenges emerges due to this huge storage of data and making operations such as querying, retrieving, and analyzing very difficult and tedious. Conventional database query methods and analytical technologies are becoming insufficient to deal with this data. Cluster analysis has become an important data analysis method which is a data mining process to unveil unknown patterns from data.

During the last few years, the mining of relational databases has become the popular research topic. Various clustering algorithms have been proposed for this type of databases, but only a few methods are proposed for the spatial databases [45]. Spatial clustering has become an active topic for researchers in spatial data mining. But spatial databases have additional requirements than other databases such as knowledge of domain prior to clustering to find out input parameters which may not be known in large databases and extraction of arbitrary shaped clusters. They aim to extract spatial patterns and features from complex data, acquiring association between spatial and non-spatial attributes and helping spatial databases to reorganize for accommodating data semantics.

An algorithm CLARANS [9] under the category of partition based methods, was proposed by Ng and Han for spatial data. Due to the high computational complexity of this algorithm for large databases storing big data, another branch of clustering methods has been discovered. Density-based methods include the clustering algorithms specifically for spatial data. These methods are able to fulfill the requirements of mining spatial data and can be able to handle very large databases. DBSCAN is the most popular algorithm under this category. But it is not able to handle large datasets and datasets of

varying density. The main problem with traditional DBSCAN algorithm is scalability and highly complex nature.

## 3.2 Problem Statement

Taking into consideration above stated research gaps, the problem is formulated as to propose and implement an algorithm to address the scalability and complexity issue of DBSCAN. An efficient parallel version of DBSCAN algorithm will be proposed and implemented which can handle very large datasets and reduces the execution time through parallel processing.

## 3.3 Objectives

The objective of the thesis is:

- To study and explore the various density-based clustering algorithms.
- To compare and analyze the studied algorithms on the basis of certain parameters using R-tool and Java.
- To design and implement Mapper and Reducer function for an algorithm and execution of algorithm on multimode Hadoop cluster to achieve parallel processing.
- To compare the results obtained from Hadoop, Java and R-tool.
- To show the scalability of various clustering algorithms.

## 3.4 Methodology

The methodology to be used is as under:

- To get experience with R and R-tool.
- To compare and analyze density based algorithms on R on different datasets on the basis of various parameters.
- Implementing existing algorithms in Java.
- Installing and configuring single node and multi node Hadoop cluster on Linux Mint 17.1.
- To get experience on Hadoop framework and implementation of sample problems in Map/Reduce model.
- To design Mapper and Reducer function for proposed algorithm on Hadoop framework.
- Optimizing Mapper and Reducer function to comply with large databases.
- Comparison of results obtained by implementing algorithm on Hadoop framework and Java on the basis of the execution time and clusters formed.

# Chapter 4 Design of Efficient Parallel DBSCAN algorithm

## 4.1 Existing Density Based Algorithm

As discussed in Section II, density based algorithms have acquired a huge popularity because of their application in spatial clustering. Various clustering algorithms have been proposed fulfilling the requirements of the spatial data such as DBSCAN, OPTICS, DENCLUE, VDBSCAN, GDBSCAN and DBCLASD. DBSCAN has evolved as the best representative of density-based clustering algorithms for large databases with the complexity of *O(nlogn).* DENCLUE is suitable for very large databases which cannot be handled by DBSCAN with the complexity of *O(n).*

### 4.1.1 Implementation in R tool

R-tool version 3.2.3 is used to implement algorithms. It provides various inbuilt packages which are collections of R functions, data and code. Cluster and fpc package of R provides various clustering algorithms such as K-Means, PAM, DBSCAN, etc. fpc package of R implement a simple traditional DBSCAN algorithm which takes $O(n^2)$ time. Due to the high complexity of traditional DBSCAN algorithm, an improved version of DBSCAN algorithm has been proposed using index structure which takes O(nlogn) of time. The dbscan package implements this improved version of DBSCAN and OPTICS algorithm. R also includes various inbuilt packages which provide datasets used for experiments.

**a) Implementation of traditional DBSCAN**

As discussed above, fpc[21] package implements a simple DBSCAN algorithm.

*Function*: dbscan (x, eps, MinPts=5, method=c(“hybrid, “raw”, “dist”) )

The dbscan function takes following list of arguments:

- *x*: it specifies the dataset on which clustering will be performed. It can be data matrix or data frame
- *Eps*: it is the first parameter of DBSCAN which specifies the minimum radius of a cluster
- *MinPts*: it is the second parameter of DBSCAN which specifies the density threshold for a cluster. Default value is 5

- *Method*: it tells about how data to be treated. It specifies whether data is data matrix or raw data
- There are some other arguments such as seeds. Countmode whose value can be assumed default by function.

Inbuilt package *datasets* provide a list of datasets from which iris dataset and USArrests is be used. Dataset Animals is used from *cluster* [23] package and wine dataset is used from *rattle* [32] package. Table 4.1 shows the information about datasets including packages which provide datasets, the number of attributes of particular datasets and instances in a dataset.

Table 4.1: Information about datasets

| ***Packages in R*** | ***Datasets*** | ***#Attributes*** | ***#Instances*** |
|---|---|---|---|
| Cluster | Animals | 6 | 20 |
| Datasets | USArrests | 4 | 50 |
| Datasets | Iris | 5 | 150 |
| Rattle | Wine | 14 | 178 |
| Datasets | Quakes | 5 | 1000 |

Dbscan on iris dataset with parameters eps=0.4 and MinPts=3

Figure 4.1 shows the all the dataset provided by inbuilt package *datasets* in R.


Fig 4.1: Datasets provided by package *dataset.*

For execution of commands in R, first packages to be used should be imported. In this algorithm, we will be using two packages: *datasets* and *fpc*. Iris datasets has 5 attributes in which 5$^{th}$ attribute is categorical on which clustering cannot be performed. So, it is to be ignored for running dbscan function. For this, matrix is constructed with first 4 attributes of iris data. Parameters applied for execution are: eps=0.4 and minpts= 4. Execution of commands is illustrated in figure 4.2.

```
> library(fpc)
> x<-as.matrix(iris[,1:4])
> db<-dbscan(x,eps=0.4,MinPts=4)
> db
dbscan Pts=150 MinPts=4 eps=0.4
        0  1  2  3 4
border 25  4  7  7 3
seed    0 43 31 29 1
total  25 47 38 36 4
```

Figure 4.2 Execution of dbscan function in R

It shows that dbscan on iris datasets extract 4 clusters 1,2,3,4 with 0 showing as noise points.

For the graphical representation of clusters, *ggplot2*package is used. First *ggplot2* package is loaded in R and data table is constructed categorized by cluster obtained and new attribute *level* is added which stores the cluster id for each of the data point. Hulls define the polygon to be constructed for a particular cluster. The execution of commands is illustrated in Figure 4.3 and the output obtained of the execution of this command is shown in figure 4.4. In figure 4.4, red, blue, green and yellow polygons portraits four clusters obtained and noise points are shown in black.

```
> library(ggplot2)
> dt<-data.table(iris,level=as.factor(db$cluster),key="level")
> hulls <- dt[, .SD[chull(Sepal.Length,Sepal.Width)], by = level]
> hulls <- hulls[level != "0",]
> ggplot(dt,aes(Sepal.Length,Sepal.Width,color=level))+
+ geom_point() +
+ geom_polygon(data=hulls,aes(fill=level,group=level),alpha=0.2,color=NA)+
+ scale_color_manual(values=c("red","blue","green","yellow","black"))+
+ scale_fill_manual(values=c("red","blue","green","yellow","black"))
```

Fig 4.3: Execution of ggplot2 in R

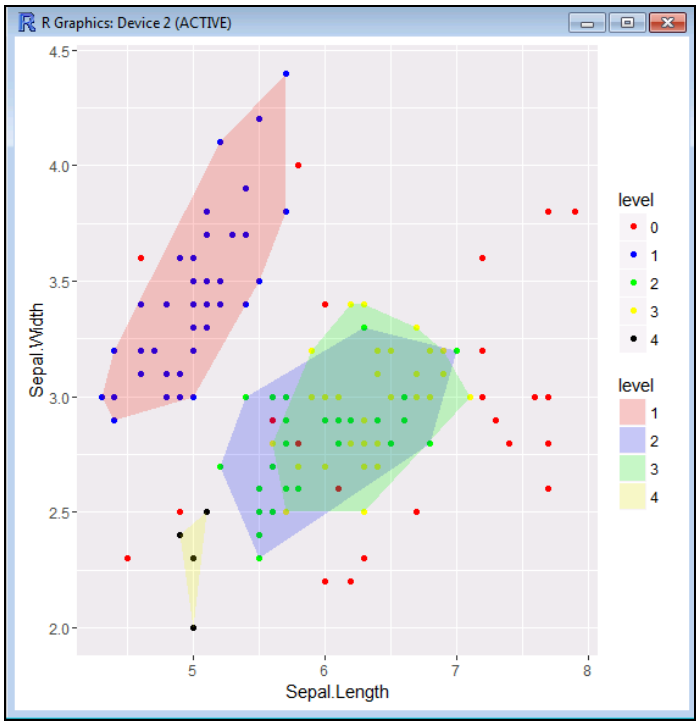


Fig 4.4: Clusters obtained in iris data

**b) Implementation of improved version of DBSCAN**

As discussed before, dbscan [20] package implements an improved version of DBSCAN algorithm. It re-implements DBSCAN clustering algorithm using an indexing structure: kd trees. This is faster and can handle large datasets than that of DBSCAN implemented by fpc.

*Function*: dbscan (x, eps, MinPts=5, search="kd tree")

Dbscan function takes following arguments:

- *x*: it specifies dataset for clustering. It can be data matrix or data object
- *Eps*: it specifies the radius of the neighborhood for a data point
- *minPts:* it specifies the density threshold which takes 5 as a default value
- *Search*: it specifies what type of search need to be made to find out the neighborhood. It can be kd, linear or dist based
- There are other arguments such as split-rule, bucket-size which can take default values.

In this, *dbscan* package is loaded and same method applied in iris data as applied in execution of dbscan function in fpc package. Parameter values used are: eps=0.4 and minPts=4. Figure 4.5 shows the commands executed in R.

```
> library(dbscan)
Warning message:
package 'dbscan' was built under R version 3.3.0
> x<-as.matrix(iris[,1:4])
> db<-dbscan(x,eps=0.4,minPts=4)
> db
DBSCAN clustering for 150 objects.
Parameters: eps = 0.4, minPts = 4
The clustering contains 4 cluster(s).
Available fields: cluster, eps, minPts
```

Fig 4.5: Execution of dbscan function in R

It shows that it forms 4 clusters and also it provides an additional field to find out which point lies in which cluster. Figure 4.6 shows the points lying in which cluster. Here, 0 denotes noise points which are not in any cluster.

```
> db$cluster
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 4 2 2 4 2 0 2 0 2 2 2 0 2 3 2 3 2
 [75] 2 2 2 2 2 2 2 2 2 3 2 0 2 0 2 2 2 2 2 4 2 2 2 2 4 2 0 3 3 3 3 0 0 0 0 0 3
[112] 3 3 3 0 3 3 0 0 0 3 3 0 3 3 0 3 3 3 0 0 0 3 3 0 0 3 3 3 3 3 3 3 3 3 3 3
[149] 3 3
```

Fig. 4.6: Points with their cluster id

**c) Implementation of OPTICS**

The dbscan package also implements another popular density based clustering algorithm OPTICS using kd trees.

*Function*: dbscan (x, eps, MinPts=5, eps_cl, search="kd-tree")

Optics function takes following parameters:

- *x*: dataset for clustering which can be data matrix or distance matrix
- *Eps*: it is the generating distance of OPTICS algorithm which is the upper bound on the radius of the neighborhood
- *minPts*: it specifies the density threshold with default value:5
- *eps_cl*: it is the reachability distance of OPTICS algorithm which is the threshold value to discover clusters
- There are other parameters which can take a default value.

OPTICS algorithm is implemented on iris data using parameter value as: eps=01, minPts=4 and eps_cl=0.4.Figure 4.7 shows the commands executed in R.

```
> library(dbscan)
> x<-as.matrix(iris[,1:4])
> opt <- optics(x, eps = 1, minPts = 4, eps_cl = .4)
```

Fig 4.7: Execution of optics function in R

Output of OPTICS algorithm is a reachability plot which shows the ordering of the points. Function *plo*t is used to show the reachability plot as shown in figure 4.8.

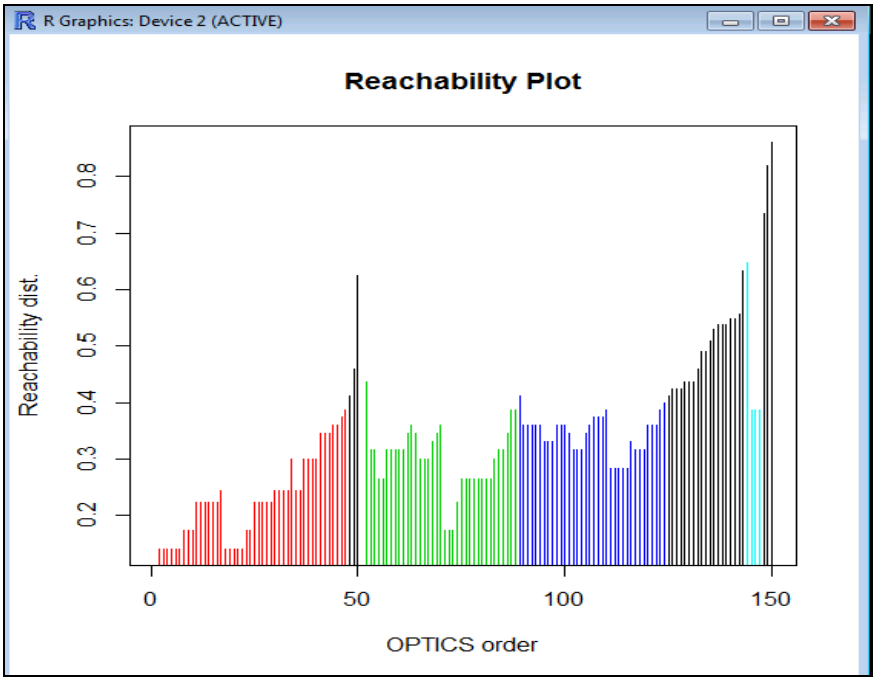


Fig 4.8: Reachability Plot for iris data

Besides iris data, fpc:DBSCAN, dbscan:DBSCAN and dbscan:OPTICS was also implemented on other four datasets described in table 1. They are compared on the basis of different parameters and graph is obtained.

### 4.1.2 Implementation in Java

After analyzing two versions of DBSCAN [11] and OPTICS [10] algorithm, the best-suited algorithm is chosen according to the required requirements. A traditional DBSCAN and an improved DBSCAN are implemented in Java using Eclipse IDE and their run-time is compared. In DBSCAN, the most important step is to find neighbors of a point, known as RegionQuery . For this step, the entire dataset is required to scan for each of the data points which take $O(n^2)$ time. In traditional DBSCAN, RegionQuery is implemented using simple search technique which scans full dataset. In an improved version of DBSCAN, kd tree is constructed for the dataset which takes *O(nlogn)* time and searching process in this tree takes *O(nlogn)* of time.

**I. Implementation of traditional DBSCAN: DBSCAN-S**

DBSCAN algorithm is implemented in java using three classes. *Point* class is used to initialize the x-y coordinates of the points. *Utility* class has the implementation of main function of DBSCAN such as finding key point, merging clusters. Arraylist of point type stores the input points and Arraylist of list of points stores the output of an algorithm. Parameter values are initialized by a user. To find out the distance between two points, Euclidean distance is used. First, it finds the clusters depending only upon the condition of core points. For each of the point, core point condition is checked and if neighbors of a point consist of more than MinPts then it forms a cluster. To merge these clusters, it uses the concept of "sharing object" which must be a border point. If two clusters obtained from core points share an object, then, they can be merged. Algorithm DBSCAN is defined as Algorithm 4.1.

**Input:** Eps, MinPts, Input file

**Output:** ArrayList of List of points storing clusters of points.

**Prerequisites**: Construct **Point Class** which initializes x and y coordinates of input dataset. Construct **Utility class** which consists of function to fetch input file, to find the Euclidean distance between two points, to check the condition of Core Point and to merge the two cluster list obtained from core point condition.

**DBSCAN-S (Eps,MinPts)**

1. Initialize parameters.
2. For each of the data points in data set, for point p = 1 to N where N is size of *pointslist* which is an input list.
    a. If point "p" is not classified, then,
        i. New arraylist is initialized to store the neighbor points.
        ii. *IsKeyPoint* function of Utility class is called to find if "p" is core point or not. It returns the arraylist storing neighbor points of "p" if it is a core point.
        iii. If arraylist returned is not empty, then, cluster is obtained and all the points of arraylist are classified by calling the function *SetListClassified* and added to output list *resultlist*.
    b. Fetch next point from database and goto step 2.
3. Cluster from core points are obtained till this step.
4. Iterate over output list *resultlist*.
    a. Fetch two lists from it and call *MergeLis*t function of Utility class.
    b. If two lists share any object, then, these lists are merged.
    c. Second list is cleared from output list.
5. Output the final arraylist *resultilist* storing the cluster of points

Algorithm 4.1: DBSCAN-S

To implement this algorithm, following functions are used which are defined in Utility class.

- *getDistance( )*: it finds the Euclidean distance between two data points.it takes two arguments which are two points in which distance to be find. It gets x and y coordinate of points using *getX( )* and *getY( )*function of Point class and return distance
- *isKeyPoint( ):* It finds the neighbors of a point and check if it is core point.The input to this function is pointslist which stores all the points, point *p*, value of eps and minpts. For each of the point in pointslist, it calls

the function *getDistance( )* to find the distance between points and check if it less than eps value. If it satisfies this condition, then, neighbor point is added to list. If list contains points more than that of minpts, then, list is returned otherwise, null is returned

- *setListClassed( ):* Input to this function is arraylist. It classified all the point of list by calling the function *isClassed( )* of point class.
- *Mergelist( ):* Input to this function is two lists to be merged. It checks if these list share any object. If they share, then, points of second list which are not in first list, are added to first list
- *getPointsList( ):* It reads file from computer location using buffered reader and add points to list and returns this list of points.

This algorithm is tested on the file consisting of 100 points. Time taken by an algorithm for execution is 77 milliseconds. It forms 2 clusters with first cluster consisting of 55 points as shown in figure 4.9 and second cluster consisting of 45 points as shown in figure 4.10.

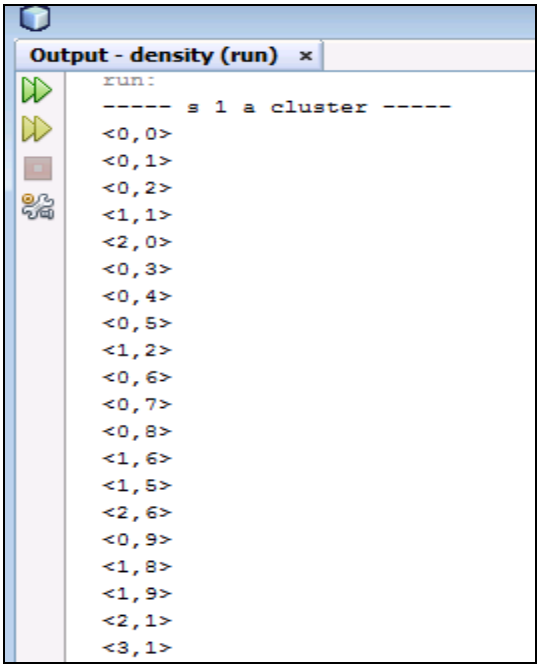


Fig 4.9: Output of DBSCAN in java: first cluster

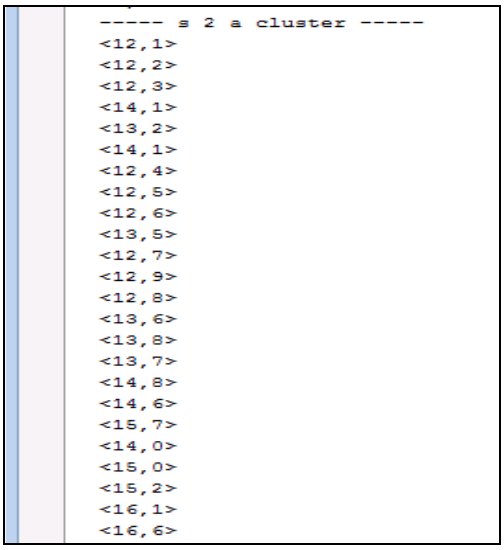


Fig 4.10: Output of DBSCAN in java: second cluster

## II. Implementation of DBSCAN using kd trees: DBSCAN-KD

This algorithm uses indexing structure kd-trees for effective search of neighbor points from the dataset. It uses the jar file which consists of declarations and definitions of functions used for implementation of kd-trees. An implementation of DBSCAN algorithm requires three classes*: Point* class, *Cluster* class and *Dbscan* class which is main class. Point class is used to implement functions which initializes x and y coordinates of point and find Euclidean distance between two points. Utility class finds the neighbors points of a cluster and expands it.

An algorithm works in two steps: first, it forms clusters from core points and then, from border points. It starts by constructing kd-trees of the data points and uses "nearest" function of kd-trees to find the neighbor points from the dataset. Then, it classifies points into three categories with the help of user-defined function getNearestNeighbor. After classification, it builds clusters from the core points and from the border points in two steps. It outputs *k* number of files consisting of points in a particular cluster where *k* is the number of clusters extracted by algorithm.

**Input**: Path to input file, Eps, Minpts

**Output**: k files consisting of points and another file for outliers.

**Prerequisites:** Construct **Point Class** which set the value of x and y coordinates of points and find the Euclidean distance between two points.

Construct **Cluster Class** which finds the size of cluster and checks if point lies in the Eps neighborhood of all points of a cluster.

**DBSCAN-KD (Datafile, Eps, Minpts)**

1. Initialize parameter values and set path to input file.
2. Read data points using buffered reader by calling function *readDataPoints( )* and stores in array of Point class *dataPoints*.
3. *initializeKDTree( )*: For each of the point in *dataPoints*, i.e., for point 1 to N where N is size of *dataPoints*
    a. Insert point in kd-tree by calling inbuilt function *insert*( ) of kd trees.
4. *classifyPoints( ):* for each of the point: 1 to N
    a. Get nearest neighbors of point *p* using function *getNearestNeighbours( )* and store in object array *NearestPoints.*
    b. Initialize *SatisfyingPoints*: 0
    c. For each of the point *q* in *NearestPoints*
        i. Find the distance between point “p” and “q”.
        ii. If(dist(p,q) < Eps) then *SatisfyingPoints*+1.
        Else If(*SatisfyingPoints* !=1) loop breaks.
    d. If SatisfyingPoints ==1, then point is Noise Point and added to arraylist *noisePoints*.
    e. Else
        i. If SatisfyingPoints == Eps, then *p* is core point and stored in arrayList *corePoints*.
        ii. Else, *p* is Border Point and stored in arrayList *borderPoints.*

5. *buildClustersFromCorePoints( ):*
    i. for each of the point *p* in *corePoints*
    ii. New cluster is created and point is added to it by calling function *createNewClusterAndAddPoint( )*
    iii. For each of newly cluster created, id of each cluster is added to arraylist *clusters.*
    *iv.* For each of the index *i* in *clusters*
        1. Call function *isInEPSProximityFromAllPoints( ):* it add points to cluster *i,* if it lies in eps neighborhood of cluster.
6. *assignBorderPointsToClusters( ):*
    a. For each of the point *p* in *borderPoints*
        i. Initialize *j*: 2
        ii. Find the Nth Nearest Neighbor index of *p* in kd tree by calling function *getNthNearestNeighbour( )* and passing *j* as the value of N and fetch the point in *q*
        iii. While "q" is not core point
            1. Increment *j: j+1*
            2. Again find the index of NthNearestNeighbour for new value of *j*.
        iv. Add point *p* to cluster id of *q*.
        v. Make the cluster id of *p* and *q* same.
7. *showClusters( ):* for each of the id in *clusters*
    *a.* Get cluster in *c.*
    *b. logclusters( ):* for each of the point stored in cluster *c.*
        i. Get the first point from arraylist
        *ii.* Write the points in file using buffered writer.
    *c. logoutliers( ):* for each of the points in *noisePoints*
        *i.* Get the first point from arraylist
        *ii.* Write the point in file using buffered writer.

Algorithm 4.2: DBSCAN-KD

This algorithm uses following functions defined in Cluster class:

- *Size()*: It returns the size of cluster
- *isInEPSProximityFromAllPoints():* Arguments to this function are point of a cluster and eps. It finds whether point lies in the Eps radius of all the points of the cluster by calling function *calcDistanceFromPoint( )* of Point class.

Following functions are defined in Point Class:

- *calcDistanceFromPoint( ):* It takes point *p* as an argument and finds the Euclidean distance between two points.

This algorithm is tested on the file spaeth2_05.txt representing the locations of towns in West Germany and storing 100 points. Time taken by DBSCAN-KD is 25 milliseconds. It forms 17 clusters and consists of 53 core points and 0 noise points. Output of this algorithm in Eclipse IDE is shown in figure 4.11.

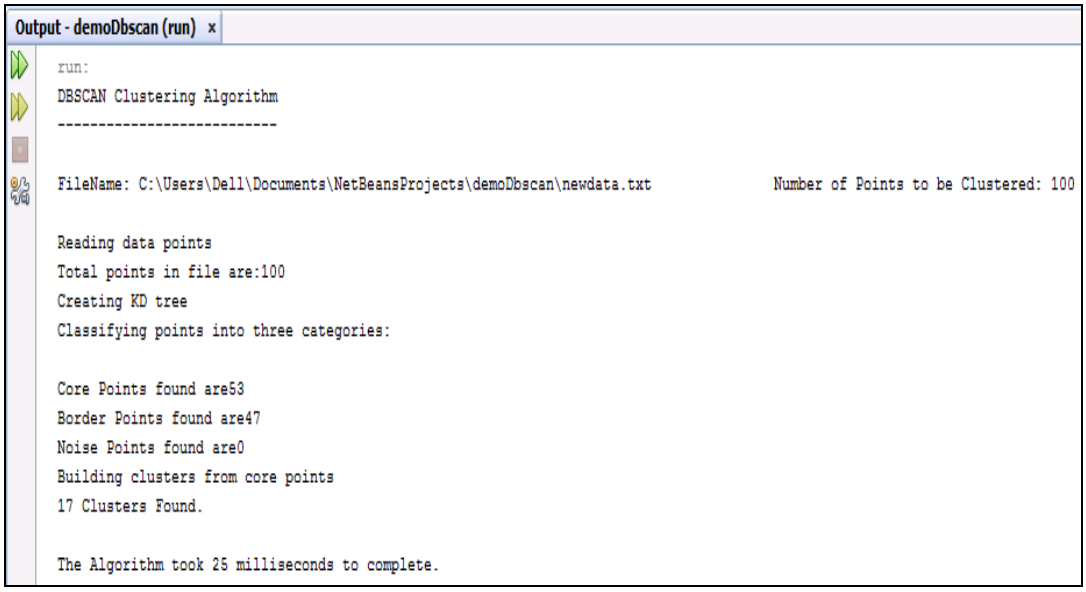


Fig 4.11: Output of DBSCAN-KD in Eclipse.

## 4.2 Proposed Algorithm: A Design of Parallel DBSCAN Clustering Algorithm based on MapReduce: MR-DBSCAN-KD

Algorithm DBSCAN discussed in 4.1 sections takes a significant amount of time when used with large datasets. But problem arises when dataset becomes very large, then, even DBSCAN-KD will also take large amount of time. To solve this inefficiency, a parallel version of DBSCAN is proposed using MapReduce programming model. It solves the problem of scalability where large dataset is partitioned into smaller parts and send to the different nodes on cloud cluster and processed in parallel manner on Hadoop framework. Both the versions of DBSCAN are implemented on single-node and multinode setup of Hadoop framework.

Some other approaches were also suggested for parallel programming model of DBSCAN. Maitery N. and Dinesh V. [33] suggested a parallel implementation of IDBSCAN algorithm which develops clusters in increments. In this algorithm, IDBSCAN will be applied on each node and clusters are obtained, then, they are merged at master node. If new data request arrives, then, data node will be responsible for further processing and forms new clusters or add points to old clusters. Vinod S. and Deepti P [34] suggested a new approach Hybrid Mechanism to parallelize the implementation of DBSCAN algorithm. This approach uses new methodology for sorting: Tera Sort on the data output by map function. X.Fu, S. Hu and Y. Wang [35] have also done research on parallel implementation of DBSCAN. In [36],[37], new algorithms were suggested to partitioning of large datasets before distributing splits to data nodes.

In the proposed algorithm, data is copied from local machine and uploaded to HDFS. Job will be submitted on Master node of Hadoop cluster and it will fetch data from HDFS and break this large data into smaller parts known as splits and send to different datanodes of Hadoop cluster for processing. Job Tracker gets the command from Task Tracker for processing the data. On each node, DBSCAN-KD algorithm will be applied and local clusters are created. After successful completion of task assigned, Job Trackers send these local clusters to Task Tracker where all clusters are merged on the basis of some criteria and global clusters are obtained. Global cluster will be obtained on Master Node which contains multiple local clusters. Both the versions of DBSCAN use the following methodology:

1. Data is uploaded on HDFS through terminal using *put* command.
2. Input data get split by Master Node and distributed to different datanodes.
3. Map function read the data from HDFS line by line and forms the <key,value> pairs of the dataset. It performs DBSCAN clustering to generate the initial local clusters. It will generate the pairs <C-id, {x,y}> after execution of map phase.
    a. Key: it stores the initial cluster-id of clusters generated locally in Mapper.
    b. Value: it is of type DoubleArrayWritable which is a user-defined data type and stores a single point of dataset. It stores the point in arraylist with the values *x* and *y* coordinate.
4. Combiner: It classifies the key value pairs with the same key into a group and distribute to idle processor for further processing. Each map function will generate large records in the form of <key,value> pairs which will consume network resources. So, combiner combines the output of map function and form the pairs in the form of <C-id, list({x,y})>. If combiner function is not designed by user, then, default combiner will be invoked by Hadoop.
5. Reduce function retrieves the key-value pair passed by Mapper and merge the local cluster on the basis of some criteria. It generates final clusters by merging the local clusters passed by Mapper.
6. This method is applied to all sites and local clusters are obtained.
7. When processing on all sites is completed, then, all Job Trackers sent their output to Task Tracker. Master node forms global clusters which is the set of multiple local clusters.

Figure 4.12 shows the flow chart of DBSCAN based on MapReduce

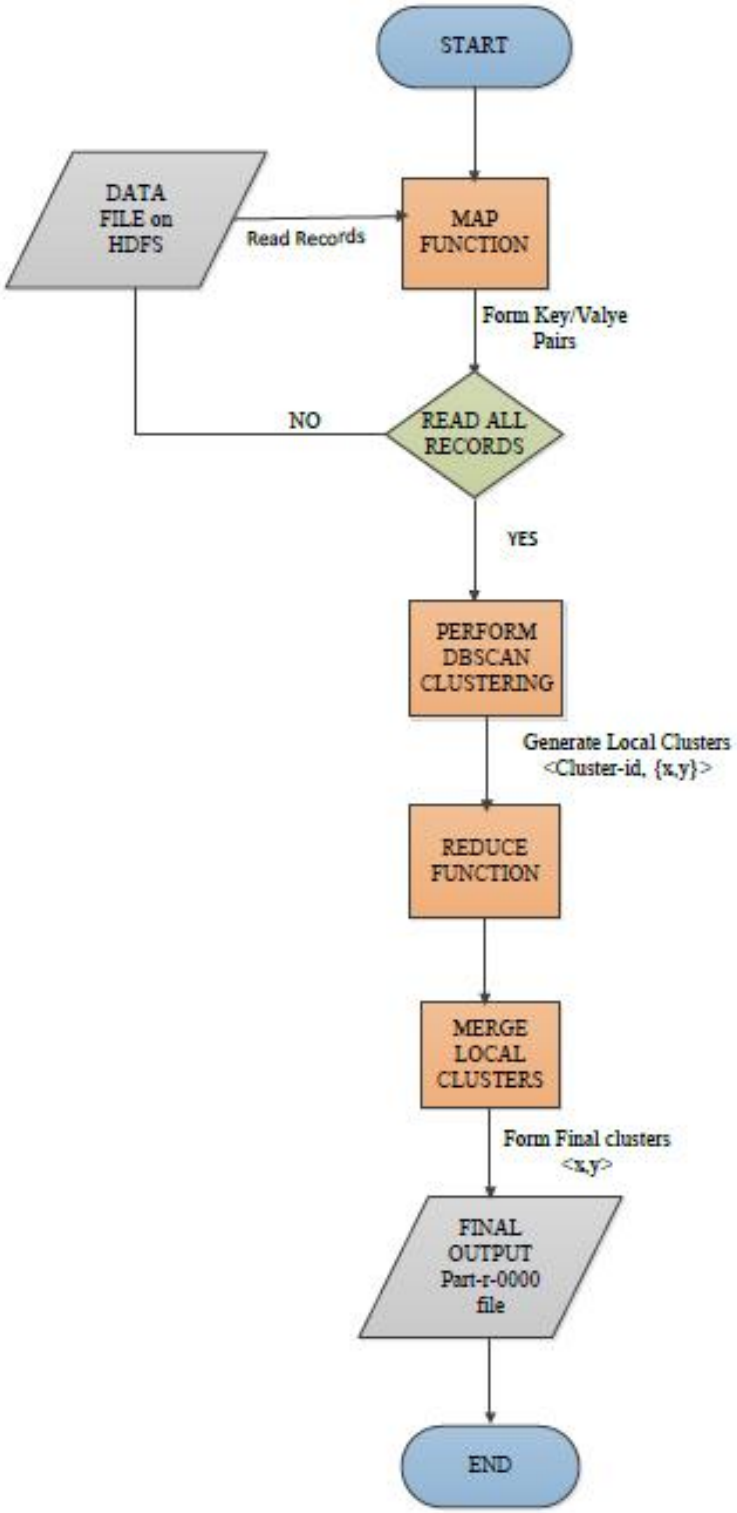


Fig 4.12: Flowchart showing flow of MR-DBSCAN

### 4.2.1 Design of parallel DBSCAN-KD algorithm: MR-DBSCAN-KD

DBSCAN is implemented with kd-trees as data structure on multinode setup of Hadoop Cluster. In Hadoop Cluster, we used one master and two slaves under common group and user. Master itself is also acting as slave. Large dataset is copied to HDFS from local machine at master site. Job is submitted to master on which processing takes place. When dataset is very large i.e., block size of data is greater than 256 KB, then, dataset get splitted into blocks and sent to the different nodes. Map function process the input file, forms <key,value> pairs and perform DBSCAN algorithm and generates clusters and send them to reducer. Reducer merges the clusters generated in Mapper to form global clusters. Output of Reducer will be stored in HDFS in a file.

DBSCAN-KD algorithm is implemented in Map/Reduce programming model. Methodology used by MT-DBSCAN-KD is defined as follows:

- In map function, data file is read by Mapper line by line and forms key/value pairs. Data file consists of points in the form *(a,b)* where *a* and *b* are of the double data type
- Input key of Mapper is of type LongWritable which stores the position of line read by Mapper
- Input value of Mapper is Text which holds one line of data file
- Map function performs DBSCAN clustering on the records retrieved from the HDFS and forms local clusters. It reads the points from the input value field and stores these points in the array of Point class type: *dataPoints*
- Map function also stores the number of lines read by Mapper in the variable *m*
- After collecting all points in the array, it calls the function to set the context where DBSCAN-KD is called to form local clusters. It forms clusters by constructing KD trees on the basis of core points by calling the function *BuildClusterFromCorepoints( )*
- Output key holds the initial cluster-id generated in the Mapper
- Output value is of the type DoubleArrayWritable which is user defined type in Hadoop. It stores the points in the form of arraylist
- In reduce function, clusters generated by Mapper are stored in an array and merges to form global clusters using the concept of Border Points by calling the function *BuildClustersFromBorderpoints( )*

- After merging all local clusters, global clusters are generated which are collection of local clusters on the Master node
- Output key and value of Reducer is of the DoubleWritable which stores the x-coordinate and y-coordinate of the points in the key and value respectively
- It initializes the variable *index* which stores the number of clusters formed by algorithm. It is written to the context and then, points in first cluster are written. For each of the cluster in arraylist *clusters*, index is incremented and written in the context before writing the points in a cluster. Output file is the part-r-0000 file which stores the output of Reducer.

**Design of algorithm MR-DBSCAN-KD:** In this algorithm, three classes are defined mainly which are **Mapper** class which has the map function, **Reducer** class which has reduce function and MRDBSCANKD class which is driver class and has main function. It also consist of all the functions defined in DBSCAN-KD algorithm except the function *readDatapoints( ).*

1. **KDMapper class:** It extends the inbuilt Mapper class provided by Hadoop library with the dataypes defined by user according to requirements.

**Input Key Datatype**: LongWritable
**Input Value Datatype**: Text
**Output Key Datatype**: LongWritable
**Output Value Datatype**: DoubleArrayWritable

**Map(LongWritable, Text, Context)**

1. Initialize the variable *m* which stores the number of lines read by Mapper.
2. Read the input line which is in Text format and stores the first line in the variable *line* which is of String type.
3. Increment the value of *m* by 1.
4. Parse the value stored in line variable by using StringTokensizer and stores the x and y-coordinate of point in the array list of type DoubleWritable *value_array*.
5. Value of this *value_array* is added to *dataPoints* array and to the *dataPointsMap* with its index
6. When all the records of the file read by Mapper, i.e., when *m* becomes equal to number of records passed by user as input parameters, then, function is called to set the context: *x(context)*
7. *x(context):*
    a. Call function of DBSCAN-KD

```
initializeKDTree()
classifyPoints()
buildClustersFromCorePoints()
```

    b. Write the generated clusters in context
        i. Output key is the cluster-id of each record and output value is the record itself.

Algorithm 4.3: Map function of MR-DBSCAN-KD

2. **KDReducer class:** It extends inbuilt Reducer class provided by Hadoop library.

   **Input Key datatype:** LongWritable

   **Input Value datatype:** DoubleArrayWritable

   **Output Key datatype:** DoubleWritable

   **Output Value datatype:** DoubleWritable

**Reduce(LongWritable *key*, DoubleArrayWritable *values*, Context)**

1. Declare variable *k* .
2. If(k <= m)
   a. For each of the unique key.
      i. Collect the clusters in the arraylist *clusters.*
3. If (k= =m)
   a. Call the function to setup context: *x(context)*
4. *x(context):*
   a. call function of DBSCAN-KD to merge clusters `assignBorderPointsToClusters()`
   b. Write the cluster points in context.
      i. Initialize the variable *index*to 0 which stores the number of clusters.
      ii. Write the value of *index* in context to specify the start of first cluster.
      iii. For each of the clusters in arraylist *clusters*
         1. Get the cluster in the variable *c* of type Cluster class.
         2. For each of the points in the cluster *c*
            a. Get the point stored in *c* in the variable *p* of the type Point class.
            b. Output key is *p.x* and output value *p.y* Context.write(p.x,p.y)
         3. Increment the value of *index* by 1.
   c. Write the noise points stored in arraylist *noisePoints*in context.
      i. For each of the point in *noisePoints*
      ii. Get the point *p* of type Point class
      iii. Write the x and y coordinate in the context

Algorithm 4.4: Reduce function of MR-DBSCAN-KD

3. **MRDBSCANKD class**: It is the driver class for the map/reduce functions which is single point of start of algorithm. It consists of main function which starts the processing of an algorithm.

**Main (String args)**

1. Call the function *initializeParameters().*
2. Set the Configuration and get the instance of Job of the configuration.
3. Set the jar using function *setJarByClass( )* by passing parameter as name of main class:MRDBSCANKD.class
4. Set the Mapper class by calling function *setMapperclass( )* and passing the name of Mapper class in the argument: KDMapper.class
5. Set the Reducer class by calling function *setReducerclass( )* and passing the name of Reducer class in the argument KDReducer.class
6. Set the datatype of output key and value of Mapper class
7. Set the datatype of output key and value of Reducer class
8. Initialize the input path to args[0] and output path to args[1]
9. Set the filesystem as of the filesystem of Configuration.
10. Wait for the job to complete and return true if it runs successfully.

Algorithm 4.5: Main function of MR-DBSCAN-KD

### 4.2.2 Design of Parallel DBSCAN-S algorithm: MR-DBSCAN-S

DBSCAN-S explained in section I of 4.1.2 is implemented on multimode Hadoop cluster. Before heading towards multinode, first this algorithm is executed on singlenode Hadoop cluster. As DBSCAN-S does not handle large datasets as compared to DBSCAN-KD, so, it is executed in Eclipse with Hadoop plugins and their run-time is compared with the simple algorithm. For large datasets, its efficiency will be checked by implementing it on multi-node.

DBSCAN-S is implemented in multinode Hadoop cluster. For this algorithm, the map and reduce function are designed and implemented. The following methodology is used by algorithm MT-DBSCAN-S:

- Map function is designed to read data from the data file and forms key/value pairs where key is of the LongWritable datatype and stores the line offset and value stores the points
- After reading all the records and storing in the arraylist *pointslist*, local clusters are generated
- Output key stores the initial cluster-id of the data point and output value stores a point in arraylist of DoubleWritable type
- These values of key and value is written in the context from which Reducer will read key and values
- Reducer function is designed to read key/value pairs send by Mapper from context and apply function to merge the global clusters generated and writes the final clusters in the context
- Output key and value of Reducer function is of DoubleWritable which stores the x and y coordinates of the points
- Output of DBSCAN-S is stored in arraylist *resultlist.* For each of the list in *resultlist*, index is set to fetched list and iterated over the points in list and written in context
- Output of MT-DBSCAN-S is file part-r-0000 which holds the points in specified in different clusters and SUCCESS file which specifies that function is executed successfully.

**Design of algorithm MR-DBSCAN-S:** In this algorithm, three classes are defined mainly which are **Mapper** class which has the map function, **Reducer** class which has reduce function and MRDBSCANS class which is driver class and has main function. It also consist of all the functions defined in DBSCAN-S algorithm except the function *getPointsList( )* defined in utility class..

1. **SMapper class:** It extends the inbuilt Mapper class provided by Hadoop library with the dataypes defined by user according to requirements.
   **Input Key Datatype**: LongWritable
   **Input Value Datatype**: Text
   **Output Key Datatype**: LongWritable
   **Output Value Datatype**: DoubleArrayWritable

**Map(LongWritable, Text, Context)**

1. Initialize the variable *m* which stores the number of lines read by Mapper.
2. Read the input line which is in Text format and stores the first line in the variable *line* which is of String type.
3. Increment the value of *m* by 1.
4. Parse the value stored in line variable by using StringTokensizer and stores the x and y-coordinate of point in the array list of type DoubleWritable value_array.
5. Value of this *value_array* is added to arraylist *pointslist*
6. When all the records of the file read by Mapper, i.e., when *m* becomes equal to number of records passed by user as input parameters, then, function is called to set the context: *x(context)*
7. *x(context):*
    a. Call function of DBSCAN-S
    `applydbscan()`
    b. Write the generated clusters in context
        i. output key is the cluster-id of each record and output value is the record itself which is datapoint.

Algorithm 4.6: Map function of MR-DBSCAN-S

2. **SReducer class:** It extends inbuilt Reducer class provided by Hadoop library. Implementation of reduce function is same as of reduce function of MR-DBSCAN-KD but last step of writing context is different in this reduce function.

**Input Key Datatype**: LongWritable

**Input Value Datatype**: DoubleArrayWritable

**Output Key Datatype**: DoubleWritable

**Output Value Datatype**: DoubleWritable

**Reduce(LongWritable *key*, DoubleArrayWritable *values*, Context)**

1. Declare variable *k* and increment it.
2. If(k <= m)
    a. For each of the unique key
        i. Collect the clusters in arraylist *resultlist.*
3. If (k= =m)
    a. Call the function to setup context: *x(context)*
4. *x(context):*
    a. call function of DBSCAN-S

```
getresult()
display()
```

5. Write the cluster points in context
    a. Initialize the variable *index*to 0 which stores the number of clusters.
    b. Write the value of *index* in context to specify the start of first cluster.
    c. For each of the list in arraylist *resultlist*
        i. If list is empty(), goto step c
        ii. For each of the points in list
            1. Get the point in the variable *p* of type Point class.
            2. Write the x and y coordinate of point in context using the function *getX()* and *getY()* of Point class. Context.write (p.getX( ), p.getY( ))
            3. Increment the value of *index* and write in context

Algorithm 4.7: Reduce function of MR-DBSCAN-S

4. **MRDBSCANS class**: It is the driver class for the map/reduce functions which is single point of start of algorithm. It consists of main function which starts the processing of algorithm.

**Main(String args)**

1. Set the Configuration and get the instance of Job of the configuration.
2. Set the jar using function *setJarByClass( )* by passing parameter as name of main class:MRDBSCANS.class
3. Set the Mapper class by calling function *setMapperclass( )* and passing the name of Mapper class in the argument: SMapper.class
4. Set the Reducer class by calling function *setReducerclass( )* and passing the name of Reducer class in the argument SReducer.class
5. Set the datatype of output key and value of Mapper class
6. Set the datatype of output key and value of Reducer class
7. Initialize the input path to args[0] and output path to args[1]
8. Set the filesystem as of the filesystem of Configuration.
9. Wait for the job to complete and return true if it runs successfully

Algorithm 4.8: Meduce function of MR-DBSCAN-S

## 5.1 Comparitive Analysis of Algorithms in R-tool

As discussed in previous chapter, two versions of DBSCAN and OPTICS algorithm are implemented in R-tool, version 3.2.3 using its various inbuilt packages. In previous section, implementation of algorithm is shown on Iris dataset. Commands executed and outputs obtained are also shown. Same methodology is used to implement an algorithm on different dataset. Datasets used for experiment are described in Table 4.1. For a particular dataset, two iterations of each algorithm are applied with different value of parameters. Parameters are chosen on the basis of preliminary analysis depending upon the clusters obtained which must be differentiable.

After the successful execution of algorithm on different datasets, comparison is made between them on the basis of various parameters. The parameters used for comparison are number of clusters obtained, execution time of algorithm, unclustered instances which specifies points which are not in any cluster and level of noise which takes values on the scale: negligible, very less, less, high, very high, average. Figure 5.1 shows the results of implementing fpc:DBSCAN and dbscan:DBSCAN on different datasets. Figure 5.2 shows the results of implementing dbscan:OPTICS on diffrenet datasets.

Table 5.1: Results of fpc:DBSCAN and dbscan:DBSCAN

| *Datasets* | *Parameters* | *Number of clusters formed* | *Runtime(in ms) dbscan: DBSCAN* | *Runtime(in ms) fpc: DBSCAN* | *Unclustered Instances (dbscan:DBSCAN)* | *Noise level (dbscan:DBSCAN)* |
|---|---|---|---|---|---|---|
| Animals | Eps=0.5 Minpts=2 | 3 | 0.243 | NA | 5 | negligible |
| | Eps=1 Minpts=2 | 1 | 0.334 | NA | 0 | 0 |
| USArrests | Eps=9 Minpts=2 | 8 | 0.276 | 11.58 | 32 | high |
| | Eps=12 Minpts=2 | 6 | 0.387 | 16.25 | 25 | less |
| Iris | Eps=0.4 Minpts=3 | 4 | 0.795 | 25.67 | 22 | very less |
| | Eps=0.6 Minpts=3 | 3 | 0.863 | 21.07 | 5 | negligible |
| Wine | Eps=6 Minpts=2 | 19 | 0.633 | 50.55 | 134 | very high |
| | Eps=7 Minpts=2 | 25 | 0.940 | 58.18 | 117 | high |
| Quakes | Eps=9 Minpts=3 | 40 | 4.15 | 340.6 | 233 | less |
| | Eps=10 Minpts=2 | 58 | 3.86 | 319.4 | 119 | less |

T able5.2: Results of dbscan:OPTICS

| *Datasets* | *Parameters* | *Number of clusters formed* | *Runtime(in ms) OPTICS* | *Unclustered Instances* | *Noise level* |
|---|---|---|---|---|---|
| Animals | Eps=0.5 Minpts=2 | 3 | NA | 5 | negligible |
| | Eps=1 Minpts=2 | 1 | NA | 0 | 0 |
| USArrests | Eps=9 Minpts=2 | 8 | 1.04 | 32 | high |
| | Eps=12 Minpts=2 | 6 | 0.63 | 25 | less |
| Iris | Eps=0.4 Minpts=3 | 4 | 1.57 | 28 | more than DBSCAN |
| | Eps=0.6 Minpts=3 | 3 | 1.74 | 5 | negligible |
| Wine | Eps=6 Minpts=2 | 19 | 1.58 | 135 | very high |
| | Eps=7 Minpts=2 | 25 | 1.64 | 117 | high |
| Quakes | Eps=9 Minpts=3 | 40 | 8.60 | 243 | more than DBSCAN |
| | Eps=10 Minpts=2 | 58 | 13.72 | 119 | less |

For comparing results obtained, a line-plot is constructed between two parameters: size of dataset and run-time of algorithm. Figure 5.3 shows the line graph. Blue line shows the execution time of dbscan: DBSCAN, red line shows the execution time of fpc: DBSCAN and green line shows the execution time of dbscan: OPTICS. As shown in graph, only dbscan: DBSCAN can form cluster with dataset of 20 instances. It is due to the fact that only DBSCAN with indexing structure support dataset containing NA values. Also, it has been deduced from graph that runtime of dbscan: DBSCAN is negligible in comparison with the execution time of fpc: DBSCAN and dbscan: OPTICS.

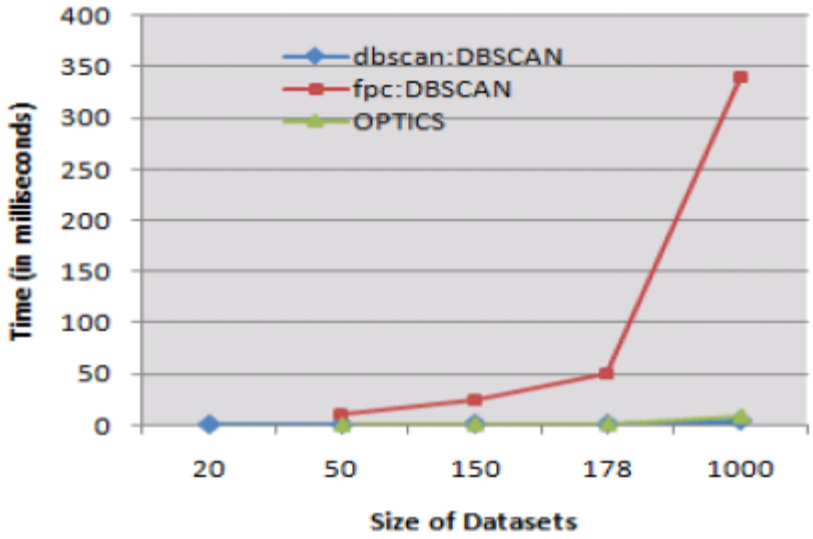


Fig. 5.1: Graph showing run-time of algorithms

**Results:**

An analysis of algorithms shows that in terms of run-time, dbscan: DBSCAN is efficient than fpc: DBSCAN and dbscan: OPTICS. Number of clusters obtained is same in all three algorithms. Compared to OPTICS and fpc: DBSCAN, execution time of dbscan: DBSCAN is negligible. OPTICS and traditional DBSCAN does not perform clustering in the data containing NA values. As shown in graph, 0 clusters are performed dataset with 20 instances which is Animals dataset provided by Cluster package. Only dbscan: DBSCAN can form clusters on Animals dataset. Execution time of fpc: DBSCAN increases exponentially with the increase in size of dataset as shown by red line in graph. From fpc: DBSCAN and dbscan: DBSCAN, dbscan: DBSCAN is more efficient in discovering clusters due to the implementation of DBSCAN with kd trees.

## 5.2 Comparative Analysis of Algorithms on Hadoop Framework

Multinode setup of Hadoop framework is used for implementing algorithm MT-DBSCAN-KD.

### 5.2.1 Implementation Environment

The proposed algorithm is developed in Java using Eclipse MARS.2 and tested by connecting three machines. The specifications of all the machines are- Intel Core i5 processor with 8 GB RAM and Linux Mint 17.3 Rebecca 64-bit Operating System. Apache Hadoop 2.6.0 framework is installed, configured and deployed on both PC servers.

### 5.2.2 Experimental Setup

All the experiments will be performed on multinode Hadoop cluster. Multinode Hadoop cluster is composed of Master-slave architecture which will be set up by connecting the multiple machines on which local Hadoop setup is tested. In figure 5.2, Cluster of two machines is shown formed by connecting two nodes. These nodes will be connected to form multi-node cluster in which one node will act as Master and others will act as Slaves. Master node will also be acting as Slave.

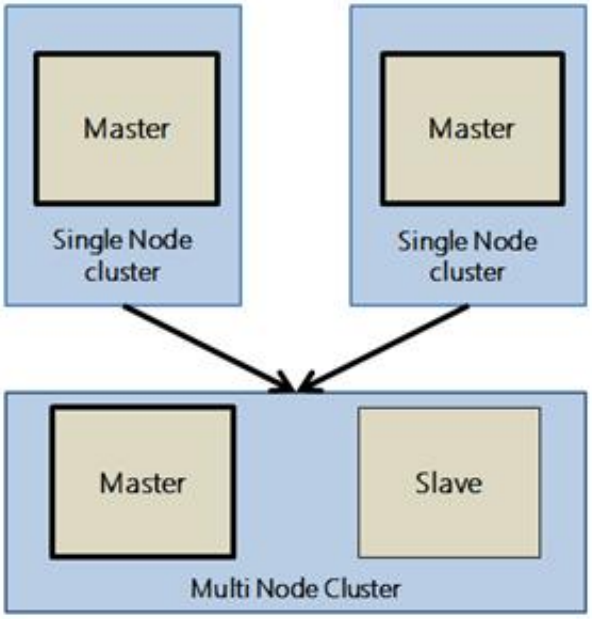


Fig 5.2 Hadoop cluster with 2 nodes (Source: www.stratapps.net)

On Master node, master daemons will be running: Namenode for the HDFS storage, Secondary Namenode, NodeManager for MapReduce processing. On both the machines, slave daemons will be running: Datanode for HDFS and Resource Manager for MapReduce. Finally, multimode setup on Hadoop cluster with 2 nodes will look as shown in figure 5.3.

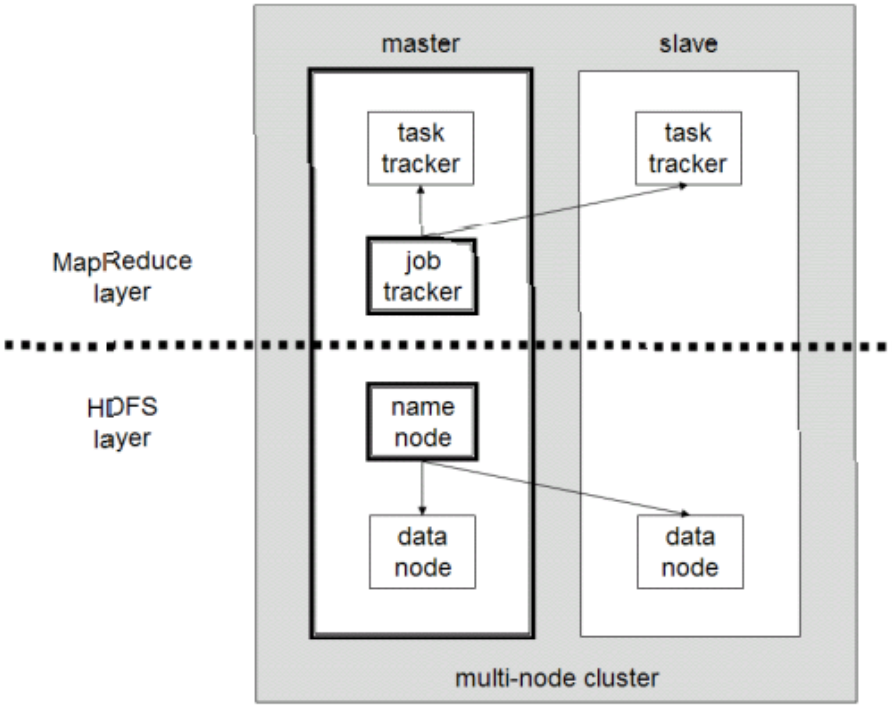


Fig 5.3: Overview of Multinode Hadoop Cluster (Source: www.quuxlabs.com)

Machines used for experiment need to be configured to setup multinode Hadoop cluster. Ipaddress of each machine will be added into the file /etc/hosts. Two machines are used for demonstration purpose whose /etc/hosts file is edited to include ip address of master and slave. Information about the host names of each machine, ip address and role played by each node is shown in table 5.3. “Master” specifies the master node of Hadoop cluster which is Namenode and JobTracker service node and “Slave-1/2/3” specifies the slave node of Hadoop cluster which is configured as Datanode and Tasktracker service. All the nodes are connected using SSH for remote access.

Table 5.3 Hadoop Cloud Computing Platform

| **Host name** | **Ip address** | **Role** |
|---|---|---|
| Master | 10.42.0.1 | Namenode & TaskTracker |
| Slave-1 | 10.42.0.11 | Datanode & JobTracker |

All commands will be directed by master node of cluster. Connection is established and cluster is started in two steps: first HDFS daemons will be started and then, MapReduce daemons. Following command will run all the daemons on both the machines.

/usr/local/hadoop/sbin/start-all.sh

Machine on which this command gets executed, namenode and secondary namenode will be run. On machine with *master* as hostname, following daemons will be run as shown in figure 5.4.

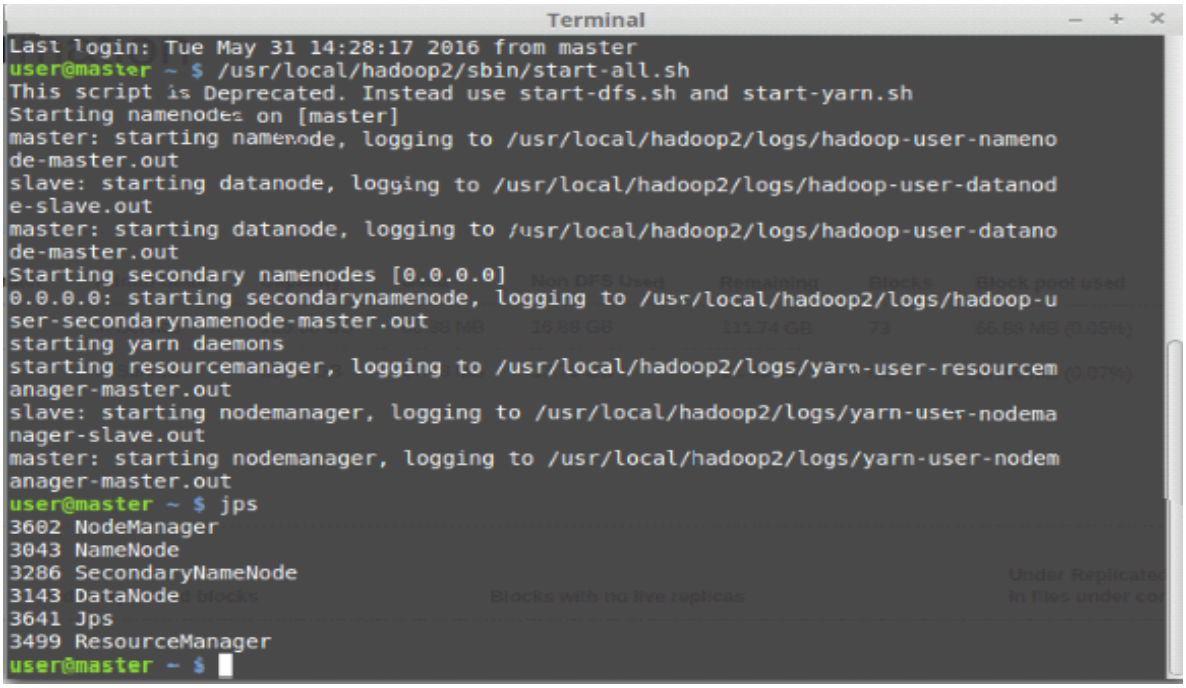


Fig 5.4 Daemons on Master node

On slave node with “Slave” as hostname, following daemons will be running as shown in figure 5.5.

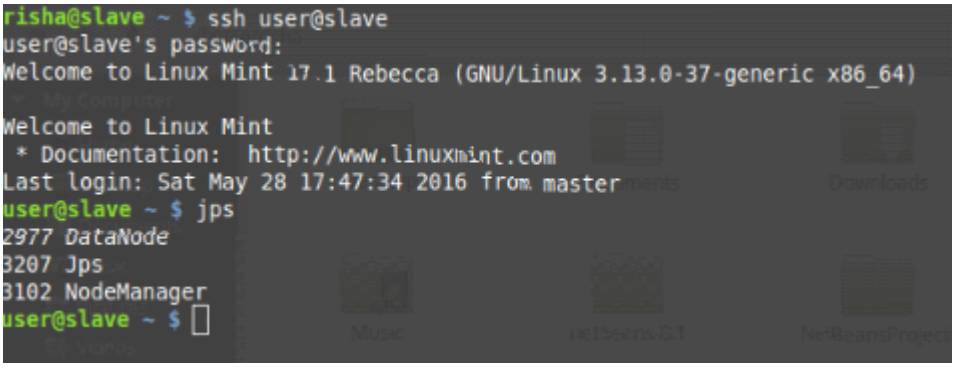


Fig 5.5 Daemons on Slave node

On machine node, namenode and secondary namenode will be running and on slave node, datanode will be running. Information about the datanode and secondary namenodes and about the HDFS space used by each node can be viewed through the webaddress *master:50070* for the namenode and datanode information as shown in figure 5.6 and 5.7 and *master:50030* for the secondary namenode information as shown in figure 5.8.

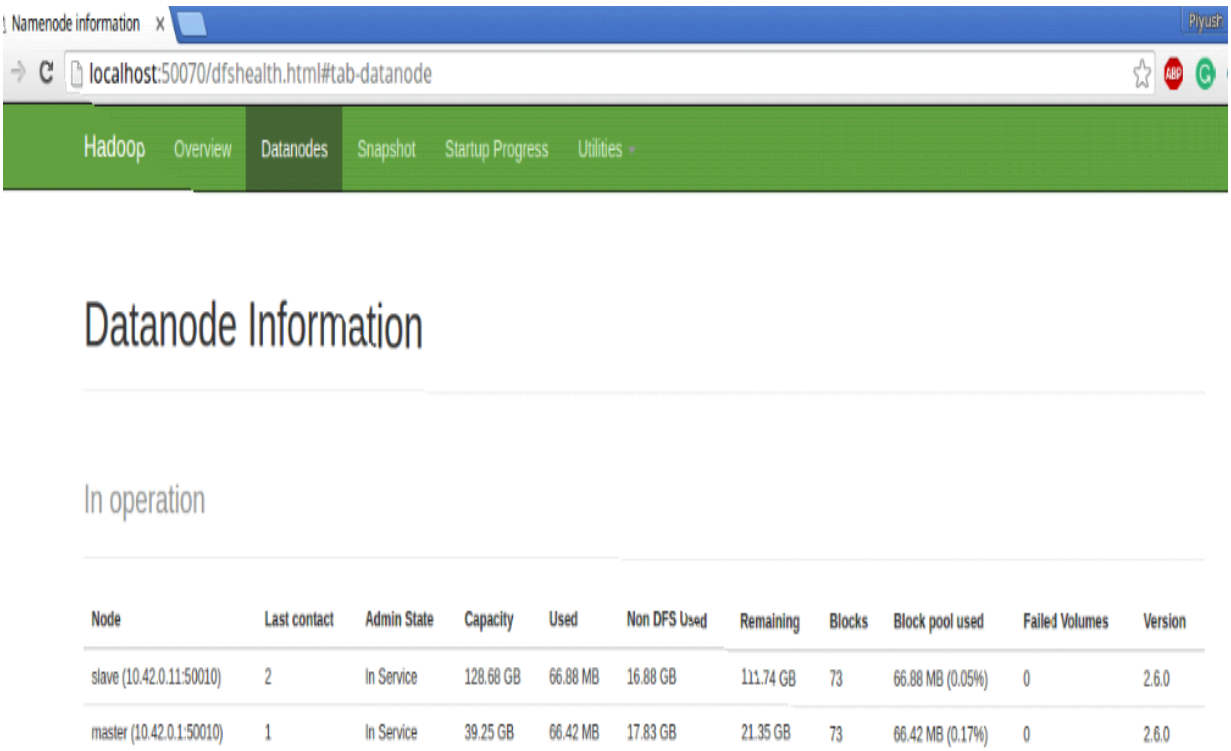


Fig 5.6 Datanode information

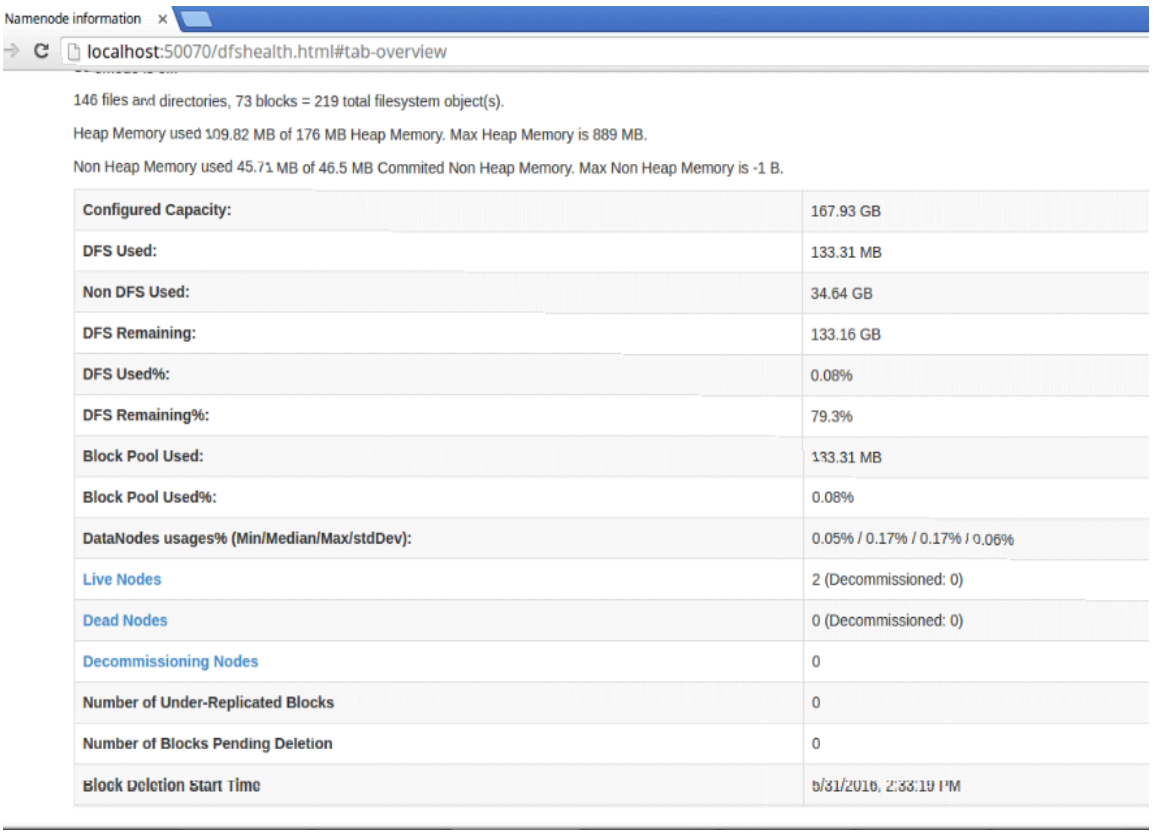


Fig 5.7 HDFS information

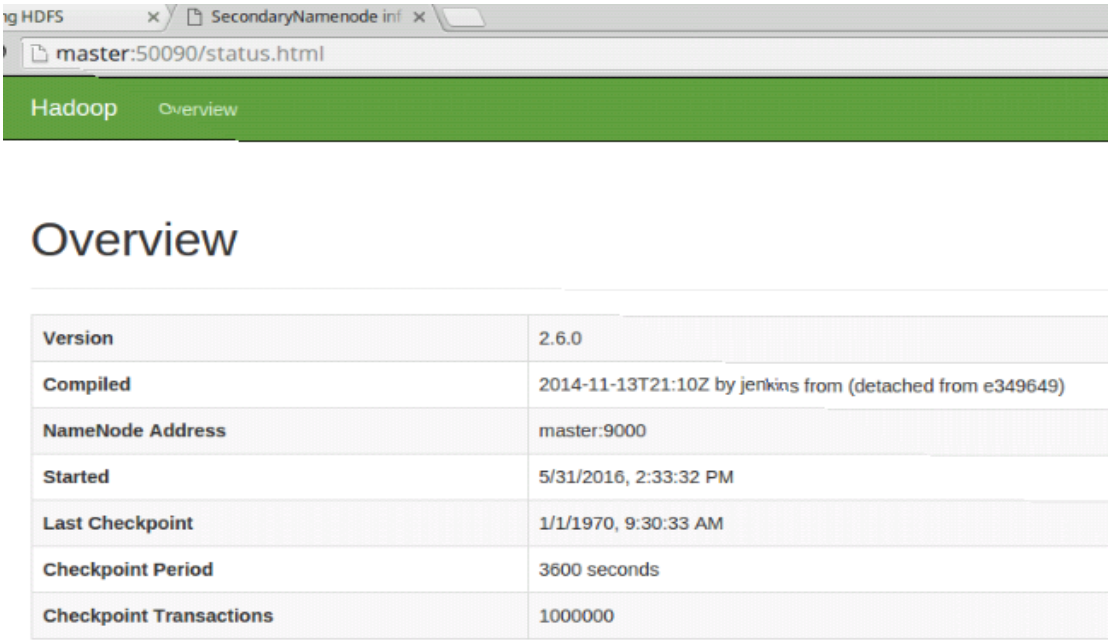


Fig 5.8 Secondary Namenode Information

For analyzing the performance of MR-DBSCAN-KD, datasets of large sizes are used for experimentation. MR-DBSSCAN-KD and MR-DBSCAN-S are implemented on Eclipse configured with Hadoop plugins and through terminal, by running the command on the jar file generated of the code to be executed on Hadoop. Information about Datasets used for experiment is shown in table 5.4 In this table, for each of the dataset, its size and number of records in a particular dataset is shown.

Table 5.4 Datasets Information used for experiment

| **Dataset** | **Size** | **Number of Records** | **Type** |
|---|---|---|---|
| Data-1 | 2.1 MB | 126000 | Folder |
| Data-2 | 27.5 MB | 1310862 | Folder |
| Data-3 | 136.43 MB | 6840000 | File |

First, datasets need to be uploaded on HDFS. This task is accomplished in steps: first, directory is made on HDFS and then, data is uploaded in the directory created. Following commands will be used for this task. Here, usr/local/hadoop/dbscandatasets directory is created where datasets will be loaded using the command *put*.

1. /usr/local/hadoop/bin/hdfs mkdir usr/local/hadoop/dbscandatasets

2. /usr/local/hadoop/bin/hdfs dfs –put /home/user/Desktop/dataset1 /usr/local/hadoop/dbscandatasets

Folders and files are uploaded in the directory dbscandatasets on the HDFS. Figure 5.9 shows all the datasets uploaded in the directory dbscandatasets and figure 5.10 shows the files in the folder "dfolder" which is Data-3 according to the table 5.4

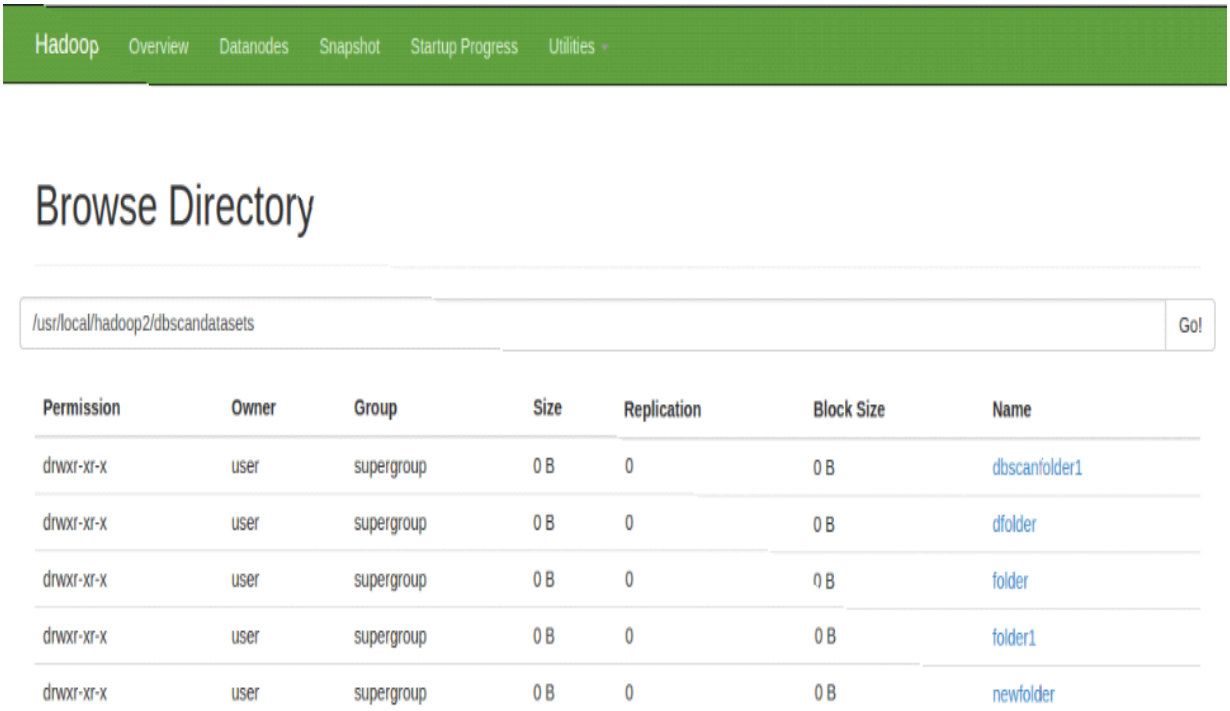


Fig 5.9 Folders in the directory dbscandatasets

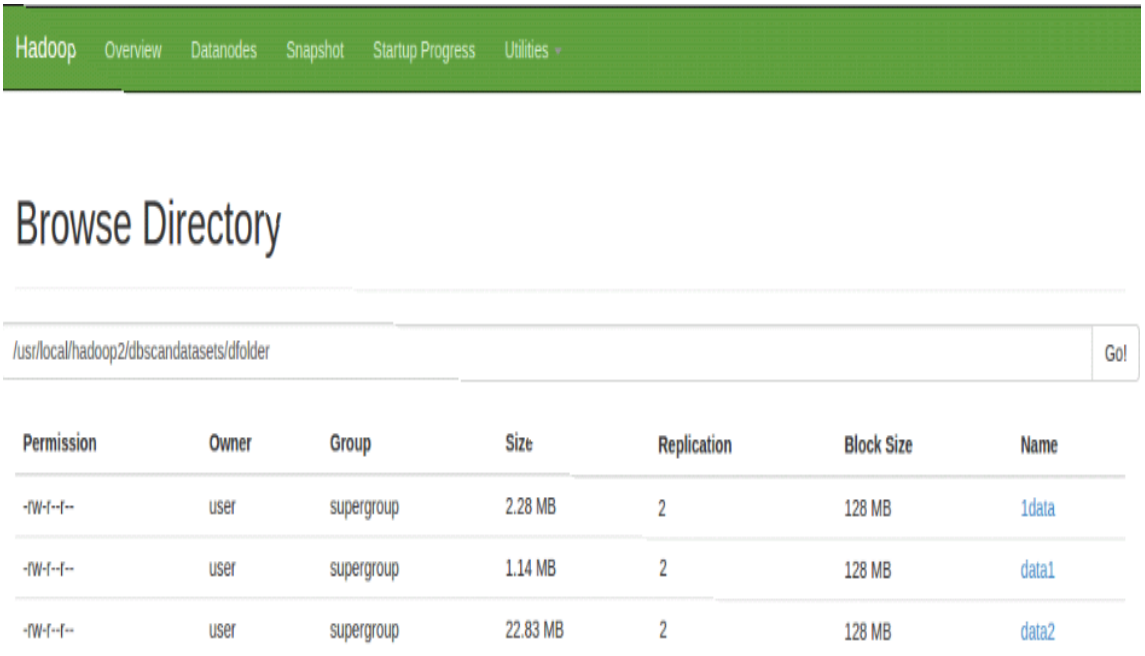


Fig 5.10 Files in the dataset "data-3"

Eclipse is configured to use plugins for Hadoop framework. With the use of plugins, HDFS locations will be directly accessible through Eclipse and MapReduce perspective can be open from Eclipse itself. Mapper and Reducer can be easily designed with these plugins as required libraries are imported automatically.

Directory created and files uploaded on HDFS through terminal with the use of commands can be visible in Eclipse as shown in figure 5.11.

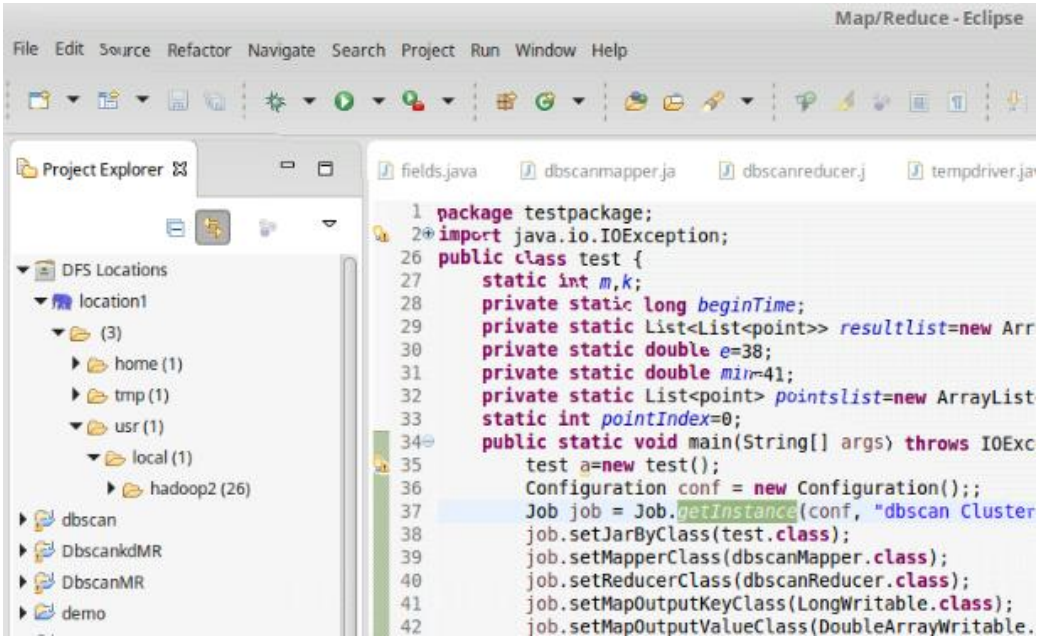


Fig 5.11 HDFS locations accessible through Eclipse

### 5.2.3 Results and Analysis

MR-DBSCAN-KD is developed in Java using Eclipse and runnable jar file is created to run it on Hadoop. Jar file is saved with the name *dbscan.jar* stored at computer location.

Command used to execute a MapReduce job:

/usr/local/hadoop/bin/hadoop jar dbscan.jar

/usr/local/hadoop/dbscandatasets/inputfilename /usr/local/hadoop2/dbscanoutputfolder

Inputfilename is the name of the input file uploaded on HDFS in the directory dbscandatasets and dbscanoutputfolder is the name of folder in which output will be stored.

1. First experiment will be informed on Data-1 which consists of three files and size of the folder is 2.1MB. Input split created for this dataset is 3.After executing job on this dataset, figure 5.12 shows the execution of the program; figure 5.13 shows the information about the file handling and figure 5.14 shows the summary of time taken by job. Total time taken by CPU for processing job is 90100 ms. Output file will be generated in the folder given in command which is dbscanoutput.

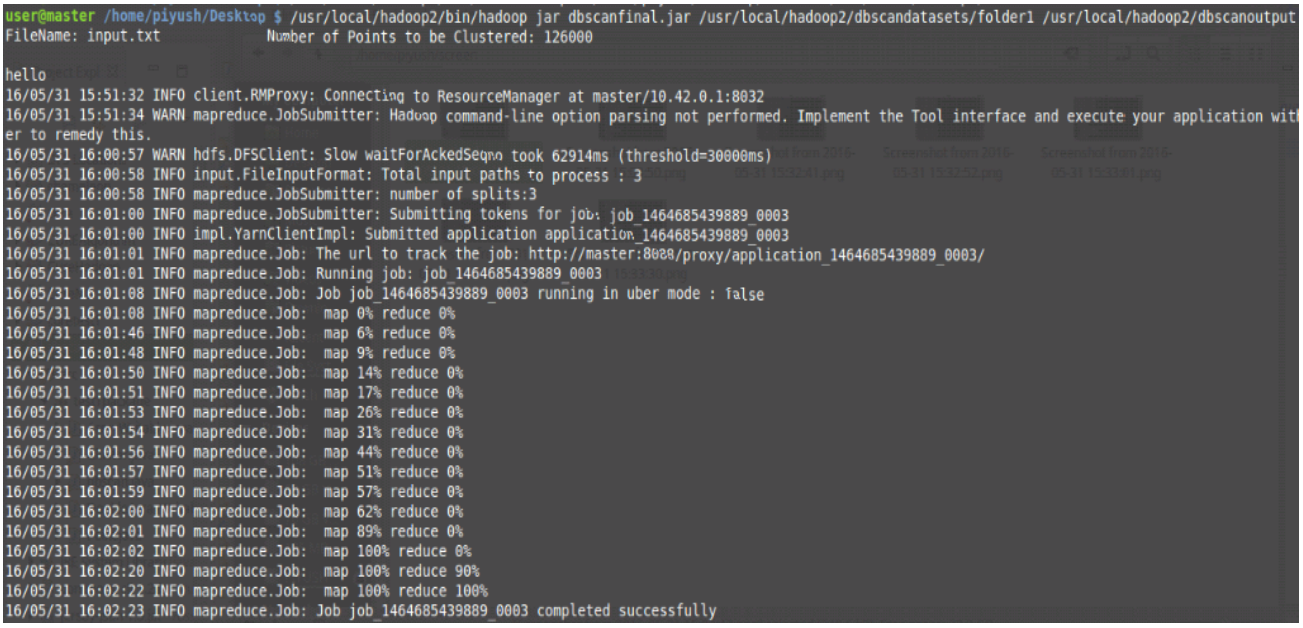


Fig 5.12 Execution flow of the program on Dataset-1

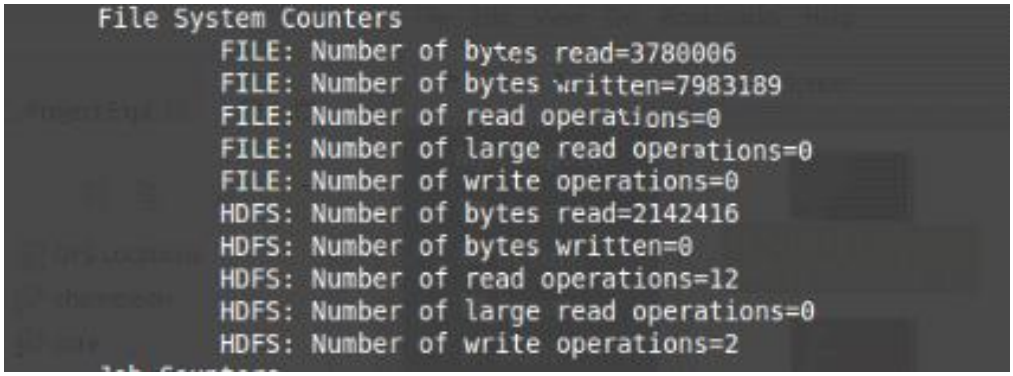


Fig 5.13 Information about file system

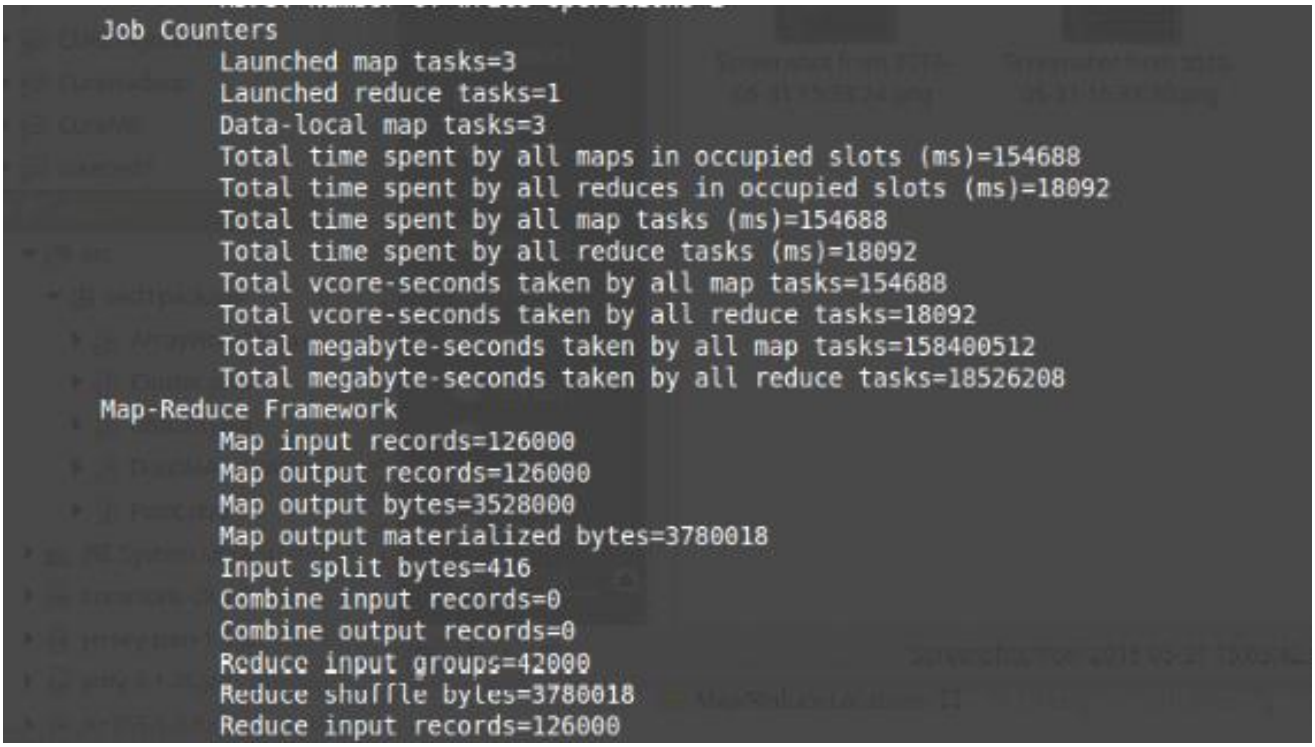


Fig 5.14 Information about job and Map Reduce framework

2. Second experiment will be carried on dataset 2 which is also a folder and consist of three files with total size as 27.5MB. An input split created for this dataset is 3. Total time taken by CPU to process this job is 173930. Output is stored in the folder dbscanoutput1 folder as given in command. Executions of command, flow of map reduce function, job information and file information is shown in figures 5.15, 5.16. 5.17

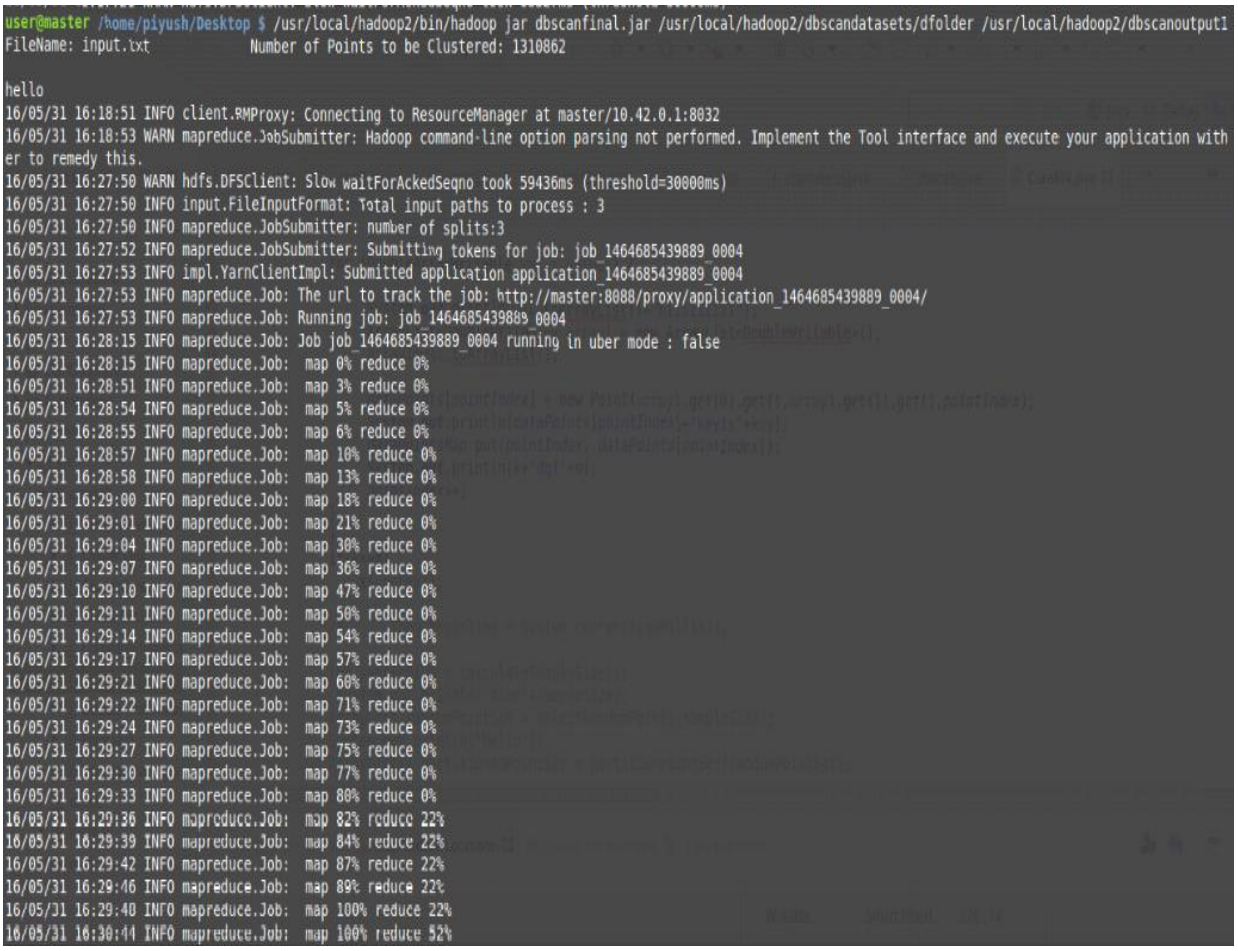


Fig 5.15 Execution of the command and flow of the program on dataset-2

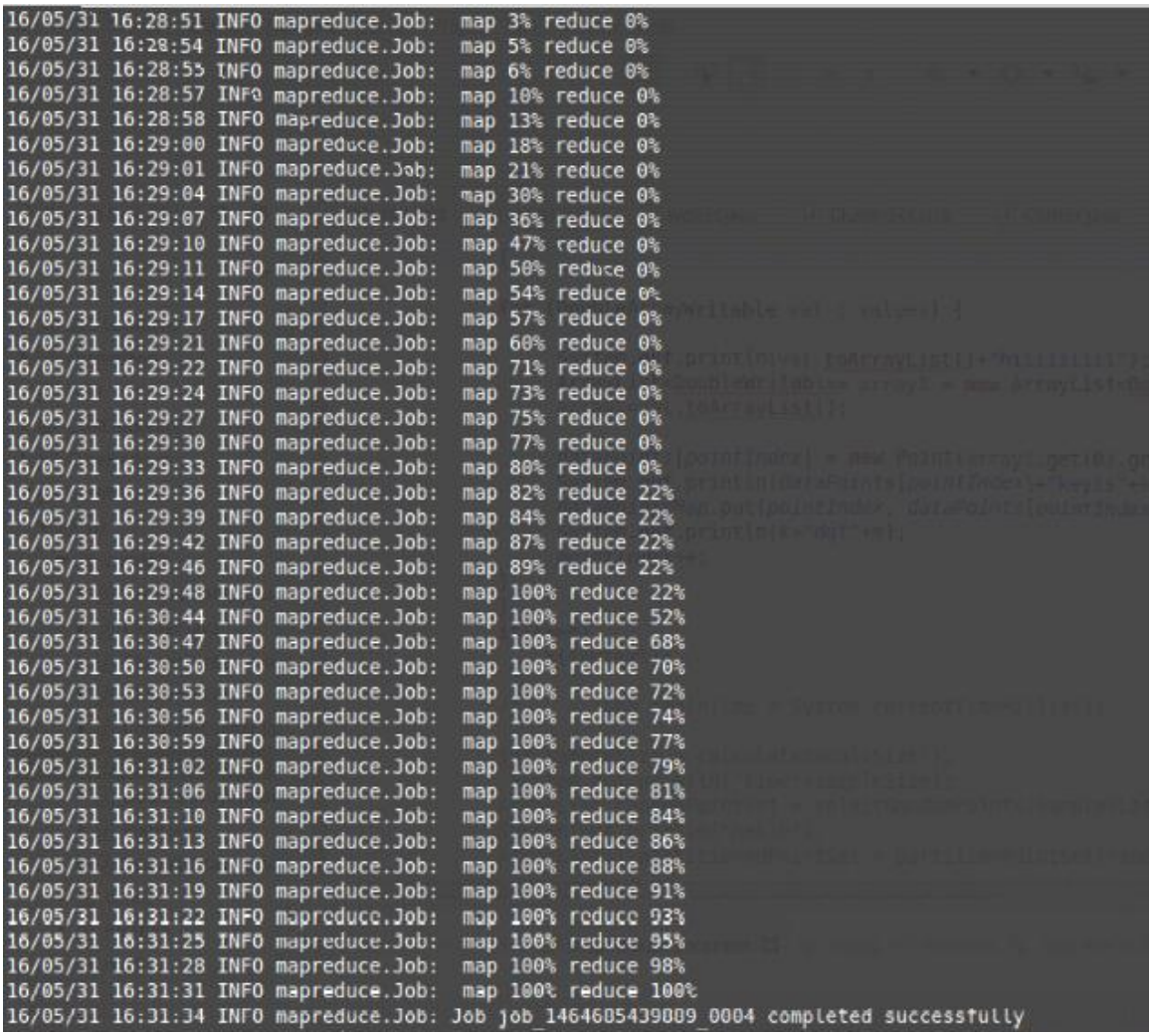


Fig 5.16 Execution flow of the job

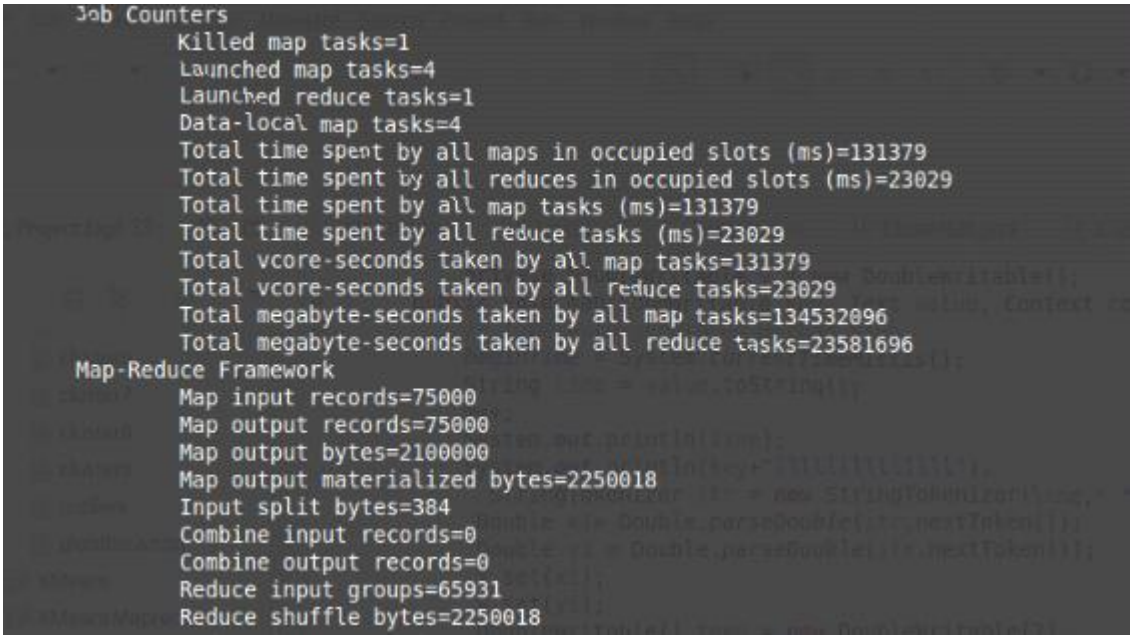


Fig 5.17 Information about job and MapReduce Framework

3. Third experiment will be carried out on the file which is of the size 136.43 MB. As the default block size in the Hadoop is 126 MB, for this file, 2 blocks will be created which will break this file. Overview of Block-1 and Block-2 created are shown in figure 5.21 and 5.22.Execution of command and flow of job is shown in figures 5.18, 5.19 and 5.20.

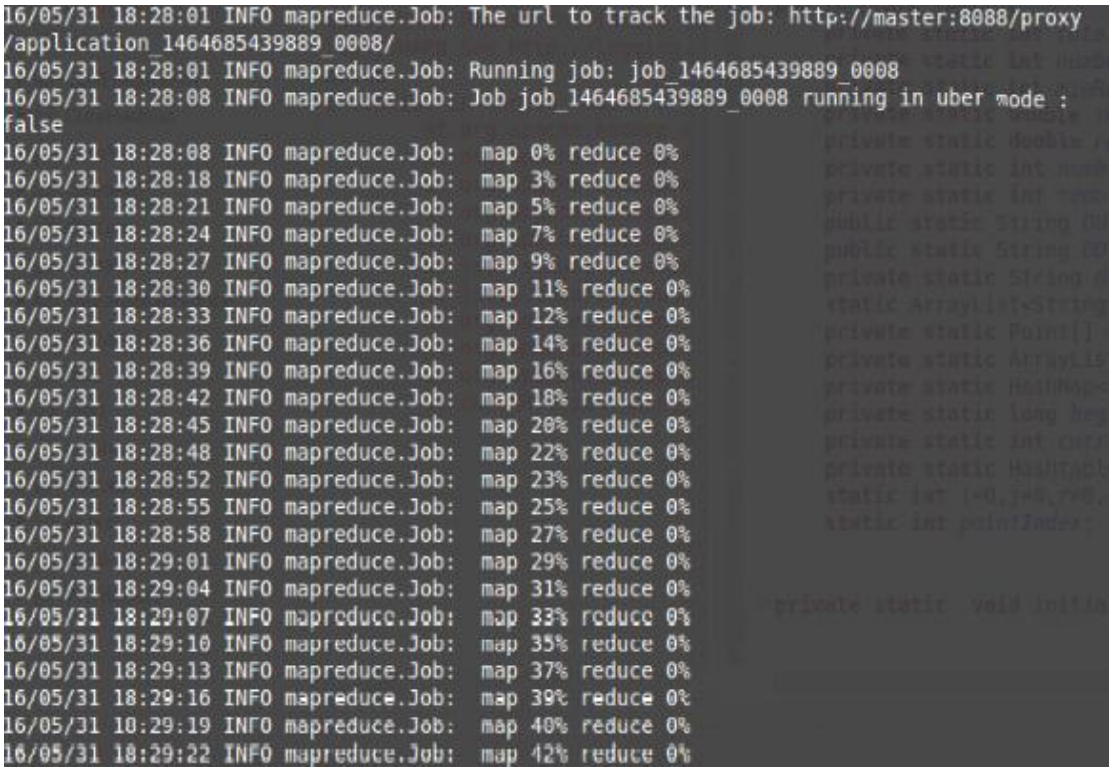


Fig 5.18 Execution flow of the job on the Dataset-3

```
16/05/31 18:29:22 INFO mapreduce.Job:  map 42% reduce 0%
16/05/31 18:29:25 INFO mapreduce.Job:  map 44% reduce 0%
16/05/31 18:29:28 INFO mapreduce.Job:  map 46% reduce 0%
16/05/31 18:29:31 INFO mapreduce.Job:  map 48% reduce 0%
16/05/31 18:29:34 INFO mapreduce.Job:  map 50% reduce 0%
16/05/31 18:29:38 INFO mapreduce.Job:  map 51% reduce 0%
16/05/31 18:29:40 INFO mapreduce.Job:  map 53% reduce 0%
16/05/31 18:29:43 INFO mapreduce.Job:  map 55% reduce 0%
16/05/31 18:29:46 INFO mapreduce.Job:  map 57% reduce 0%
16/05/31 18:29:49 INFO mapreduce.Job:  map 58% reduce 0%
16/05/31 18:29:52 INFO mapreduce.Job:  map 60% reduce 0%
16/05/31 18:29:55 INFO mapreduce.Job:  map 62% reduce 0%
16/05/31 18:29:58 INFO mapreduce.Job:  map 64% reduce 0%
16/05/31 18:30:01 INFO mapreduce.Job:  map 66% reduce 0%
16/05/31 18:30:04 INFO mapreduce.Job:  map 68% reduce 0%
16/05/31 18:30:07 INFO mapreduce.Job:  map 78% reduce 0%
16/05/31 18:30:11 INFO mapreduce.Job:  map 89% reduce 0%
16/05/31 18:30:14 INFO mapreduce.Job:  map 100% reduce 0%
16/05/31 18:34:20 INFO mapreduce.Job:  map 100% reduce 67%
16/05/31 18:34:26 INFO mapreduce.Job:  map 100% reduce 68%
16/05/31 18:34:35 INFO mapreduce.Job:  map 100% reduce 69%
16/05/31 18:34:41 INFO mapreduce.Job:  map 100% reduce 70%
16/05/31 18:34:47 INFO mapreduce.Job:  map 100% reduce 71%
16/05/31 18:34:53 INFO mapreduce.Job:  map 100% reduce 72%
16/05/31 18:34:59 INFO mapreduce.Job:  map 100% reduce 73%
16/05/31 18:35:06 INFO mapreduce.Job:  map 100% reduce 74%
16/05/31 18:35:15 INFO mapreduce.Job:  map 100% reduce 75%
16/05/31 18:35:21 INFO mapreduce.Job:  map 100% reduce 76%
16/05/31 18:35:27 INFO mapreduce.Job:  map 100% reduce 77%
16/05/31 18:35:33 INFO mapreduce.Job:  map 100% reduce 78%
16/05/31 18:35:40 INFO mapreduce.Job:  map 100% reduce 79%
16/05/31 18:35:46 INFO mapreduce.Job:  map 100% reduce 80%
16/05/31 18:35:52 INFO mapreduce.Job:  map 100% reduce 81%
16/05/31 18:35:58 INFO mapreduce.Job:  map 100% reduce 82%
16/05/31 18:36:04 INFO mapreduce.Job:  map 100% reduce 83%
16/05/31 18:36:11 INFO mapreduce.Job:  map 100% reduce 84%
16/05/31 18:36:20 INFO mapreduce.Job:  map 100% reduce 85%
16/05/31 18:36:26 INFO mapreduce.Job:  map 100% reduce 86%
16/05/31 18:36:32 INFO mapreduce.Job:  map 100% reduce 87%
16/05/31 18:36:38 INFO mapreduce.Job:  map 100% reduce 88%
16/05/31 18:36:44 INFO mapreduce.Job:  map 100% reduce 89%
16/05/31 18:36:50 INFO mapreduce.Job:  map 100% reduce 90%
```

Fig 5.19 Working of Mapper and Reducer

```
File System Counters
        FILE: Number of bytes read=410400030
        FILE: Number of bytes written=617414762
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=143639931
        HDFS: Number of bytes written=0
        HDFS: Number of read operations=6
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=123970
        Total time spent by all reduces in occupied slots (ms)=466019
        Total time spent by all map tasks (ms)=123970
        Total time spent by all reduce tasks (ms)=466019
        Total vcore-seconds taken by all map tasks=123970
        Total vcore-seconds taken by all reduce tasks=466019
        Total megabyte-seconds taken by all map tasks=126945280
        Total megabyte-seconds taken by all reduce tasks=477203456
Map-Reduce Framework
        Map input records=6840000
        Map output records=6840000
        Map output bytes=191520000
        Map output materialized bytes=205200006
        Input split bytes=111
        Combine input records=0
        Combine output records=0
```

Fig 5.20 Information about files, job and MapReduce framework


Fig 5.21 Overview of Block-1 created for dataset-3

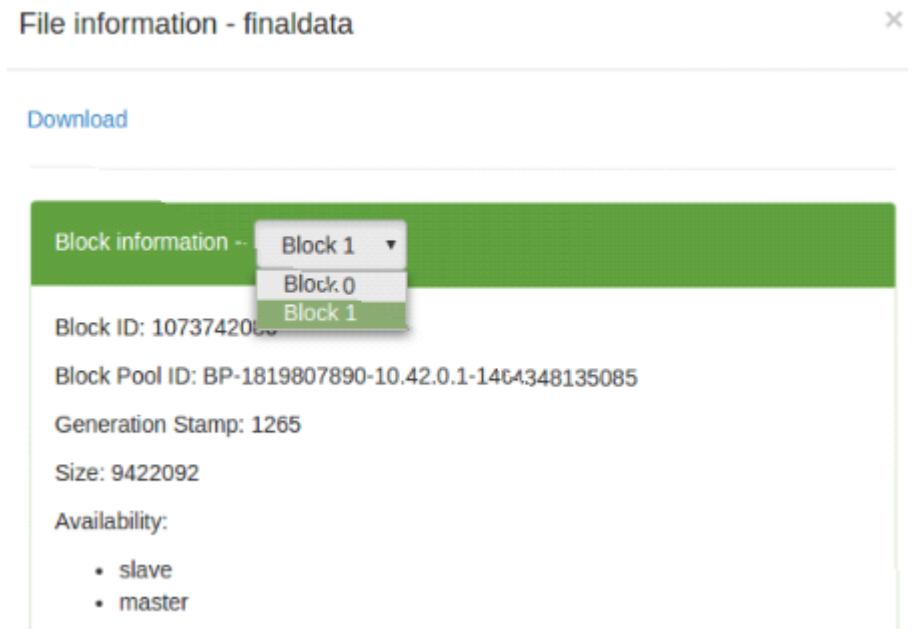


Fig 5.22 Overview of Block-2 created for dataset-3

**Comparison of MR-DBSCAN-S and MR-DBSCAN-KD:** For comparing the execution time of these algorithms, they are executed in Eclipse using Hadoop Plugins. MR-DBSCAN-S does not show any output if it is executed on datasets given in Table 5.4. So, for comparison, other datasets are used which are not very large so that output of MR-DBSCAN-S is differentiable and comparable with that of MR-DBSCAN-KD. Maximum size of dataset used is 8000 records. Output of comparison is shown in the form of graph as shown in figure 5.23. It can be deduced from the graph that both the algorithms takes same amount of time for smaller dataset but execution time increases for MR-DBSCAN-S with the increase in size of dataset as when record size is 8000, then, MR-DBSCAN-S takes greater amount of time than that of MR-DBSCAN-KD. Also, the number of clusters formed by both the algorithms varies. For very large datasets, MR-DBSCAN-S still forms very less clusters and not able to differentiate data points into meaningful clusters.

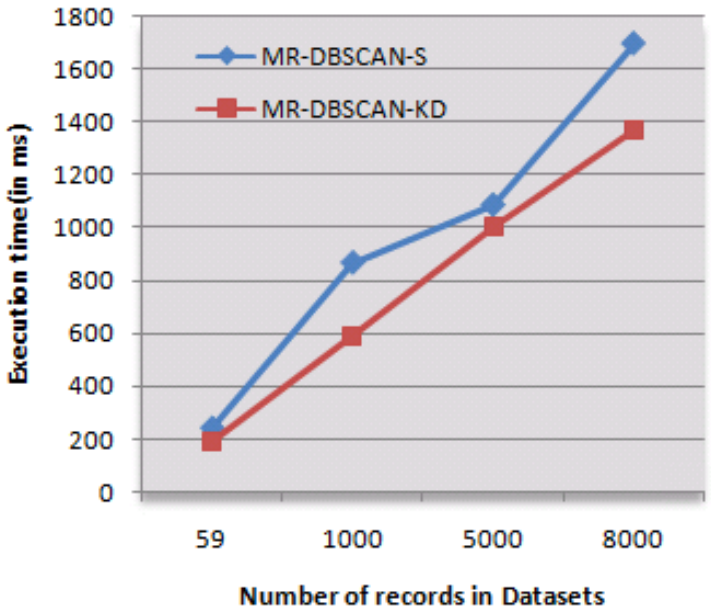


Fig 5.23 Comparison of MR-DBSCAN-S and MR-DBSCAN-KD on the basis of execution time

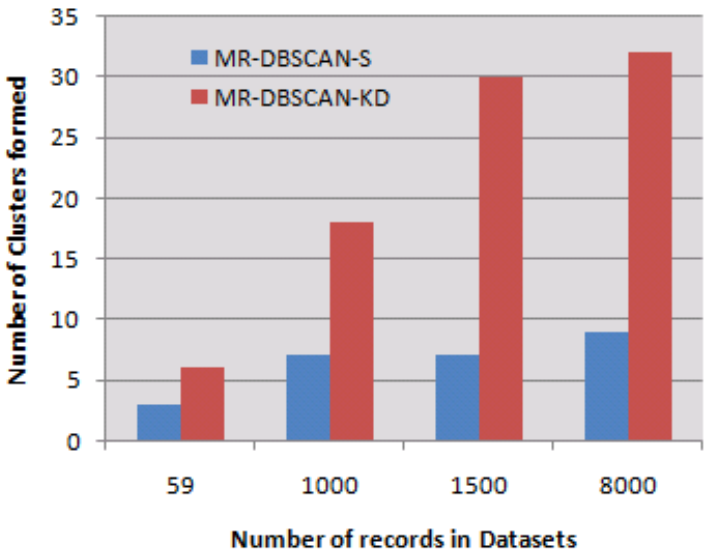


Fig 5.24 Comparison of MR-DBSCAN-S and MR-DBSCAN-KD on the basis of number of clusters formed

# Chapter 6 Conclusion and Future Scope

## 6.1 Conclusion

DBSCAN is the most popular algorithm under the category of density-based algorithms. For improving its efficiency, it is implemented with the use of indexing structure which improves its time complexity from $O(n^2)$ to O(nlogn). R tool is used for analysis of various density based algorithm which has inbuilt packages implementing both the versions of DBSCAN. An analysis has been performed using these packages on different datasets provided by R. Even an improved version of DBSCAN algorithm, DBSCAN-KD is not able to handle very large datasets and takes a considerable amount of time for these datasets. Thus, a new algorithm MR-DBSCAN-KD is proposed which is a parallel version of DBSCAN-KD algorithm implemented on Hadoop framework. It can handle very large datasets and its efficiency is also improved due to parallel processing. Mapper and Reducer function are designed for this algorithm and implemented on multinode Hadoop cluster. For the experiment, three nodes are connected where one is master and others are slave. An algorithm is tested on different datasets of maximum size of 136.43 MB which is uploaded on HDFS. A comparison is done between both the versions of DBSCAN in ECLIPSE configured with Hadoop plugins. It is observed that MR-DBSCAN-S takes a larger amount of time in comparison of MR-DBSCAN-KD when experimented on large datasets. For smaller datasets, difference in their execution time is negligible. Also, the number of clusters formed by MR-DBSCAN-S is not meaningful as it forms very less clusters even for large datasets.

## 6.2 Summary of Contributions

The contributions made by the work presented in this thesis are listed as follows:

- Survey of the various Density-Based clustering algorithms
- Comparison of the three main approaches of density-based clustering algorithms on R-tool
- Installation and configuration of Hadoop framework on local machine
- Set up of multinode Hadoop architecture by connecting multiple machines
- Design of efficient parallel DBSCAN algorithms using MapReduce

- Implementation and testing of proposed algorithm on multinode Hadoop framework
- An analysis of results obtained by implementing algorithm on Hadoop framework and Java to show scalability of proposed algorithms.

## 6.3 Future Scope

Currently, the Hadoop cluster is set up with only three nodes. It can be configured to include more nodes to validate the results accurately. Also, partitioning mechanisms can also be suggested for splitting the dataset in an effective manner to reduce interdependency between nodes and considering load balancing issue of each of the node. This mechanism can also improve the efficiency of DBSCAN algorithm to a greater extent. Further, algorithms can be tested on datasets of high-dimensions. Here, only 2-dimensional datasets are considered, so, algorithm can be used with high-dimensional datasets to verify its performance.

# References


[1] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, "From data mining to knowledge discovery in databases", AI magazine, vol. 17, no. 3, pp. 1-30, 1996.

[2] S.P. Bora, "Data mining and ware housing", In proceedings of 3rd IEEE conference on Electronics Computer Technology, pp. 1-5, 2011.

[3] T. Conollay and C. Begg, "Database Systems", 4th edition, Pearson Education Ltd, 2005.

[4] B. Minaei-Bidgoli, "Data mining for a web-based educational system", Dissertation, 2004.

[5] G.K.Gupta, "Introduction to data mining with case studies", PHI Learning Pvt. Ltd., 2014.

[6] A. K. Jain, M. N. Murty and P. J. Flynn, "Data clustering: a review", In ACM computing surveys, vol. 31, no. 3, pp. 264-323, 1999.

[7] P. Andritsos, "Data clustering techniques", Rapport technique, University of Toronto, Department of Computer Science, 2002.

[8] D.Kumar and N.Arora, "Comparative Study of Hierarchical Clustering over Partitioning Clustering Algorithm", International Journal of Innovations and Advancement in Computer Science, vol. 2, no. 4, 2014.

[9] R.T Ng and J.Han, "CLARANS: A method for clustering objects for spatial data mining", In proceedings of IEEE conference on Knowledge and Data Engineering, vol. 14, no. 5, pp. 1003-1016, 2002.

[10] M. Ankerst, M.M. Breuing, H.P. Kriegel and J. sander, "OPTICS: Ordering Points To Identify the Clustering Structure", In ACM Sigmod Record, vol. 28, no. 2, pp. 49-60, 1999.

[11] X. Xu, M. Ester, H.P. Kriegel and J. Sander, "A Distribution Based Clustering for mining in large Spatial Databases", In Proceedings of 14th IEEE International Conference on Data Engineering, pp. 324-331, 1998.

[12] A. Hinneburg and D. A. Keinn, "An efficient approach to clustering in large multimedia databases with noise", In KDD, vol. 98, pp. 58-65, 1998.

[13] P. Berkhin, "A survey of clustering data mining techniques", In Grouping multidimensional data, Springer, pp. 25-71, 2006.

[14] W. Wang, J. Yang and R. Muntz, “STING: A statistical information grid approach to spatial data mining", In VLDB, vol. 97, pp. 186-195, 1997.

[15] A. Fahad, N. Alshatri, Z. Tari and A. Alamri, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis", In proceedings of IEEE conference on Emerging Topics in Computing, vol. 2, no. 3, pp. 267-279, 2014.

[16] J.C. Bezdek, R. Ehrlicj and W.Full, “FCM:The fuzzy c-means clustering algorithm”, In Computer Geosciences, vol. 10, no. 2-3, pp. 191-203, 1984.

[17] A.P. Dempster, N.M. Laird and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm”, In Journal of the Royal Statistical Society, vol. 39, no. 1, pp. 1-38, 1977.

[18] A. Hinneburg and D. A. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering", In proceedings of 25th International conference on Very large Databases, pp. 506-517, 1999.

[19] T. Zhang, R. Ramakrishnan and M. Livny, "BIRCH: an efficient data clustering method for very large databases", In ACM Sigmod Record, vol. 25, no. 2, pp. 103-114, 1996.

[20] Package dbscan [online] Available at https://cran.r-project.org/web/packages/dbscan/dbscan.pdf

[21] Package fpc [online] Available at https://cran.r-project.org/web/packages/fpc/fpc.pdf

[22] Package microbenchmark [online] Available at https://cran.r-project.org/web/packages/microbenchmark/microbenchmark.pdf

[23] Package cluster [online] Available at https://cran.r-project.org/web/packages/cluster/cluster.pdf

[24] Package ggplot2 [online] Available at https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf

[25] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The hadoop distributed file system", In 26th IEEE Symposium on Mass Storage Systems and Technologies, pp. 1-10, 2010.

[26] J. D. Dhok, “Learning based admission control and task assignment in MapReduce”, Dissertation, 2010.

[27] H. Bäcklund, A. Hedblom and N. Neijman, "A Density-Based Spatial Clustering of Application with Noise", In Data Mining, vol. 33, pp. 11-30, 2011.

[28] G. H. Shah, C. K. Bhensdadia and A. P. Ganatra, "An empirical evaluation of density-based clustering techniques", International Journal of Soft Computing and Engineering, vol. 2, no. 1, pp. 2231-2307, 2012.

[29] X. Xu, M. Ester and H. P. Kriegel, "A distribution-based clustering algorithm for mining in large spatial databases", In Proceedings of 14th IEEE International Conference on Data Engineering, pp. 324-331, 1998.

[30] X. Xu, M. Ester, H. P. Kriegel and J. Sander, "Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications", Data mining and knowledge discovery, vol. 2, no. 2, pp. 169-194, 1998.

[31] P. Liu, D. Zhou and N. Wu, "VDBSCAN: varied density based spatial clustering of applications with noise", In Proceedings of IEEE Conference on Service Systems and Service Management, pp. 1-4, 2007.

[32] Package rattle [online] Available at: https://cran.r-project.org/web/packages/rattle/index.html

[33] M. Noticewala, Maitry and D. Vaghela, "MR-IDBSCAN: Efficient Parallel Incremental DBSCAN Algorithm using MapReduce", International Journal of Computer Applications, vol. 93, no. 4, pp. 13-17, 2014.

[34] V. S. Bawane and D. P. Theng, "Enhancing Map-Reduce Mechanism for Big Data with Density-Based Clustering", vol. 3, no. 4, 2015.

[35] X. Fu, S. Hu and Y. Wang, "Research of parallel DBSCAN clustering algorithm based on MapReduce", In International Journal of Database Theory and Application, vol. 7, no. 3, pp. 41-48, 2014.

[36] B. R. Dai and I. Lin, "Efficient map/reduce-based dbscan algorithm with optimized data partition", In 5th IEEE International Conference on Cloud Computing, pp. 59-60, 2012.

[37] Y. He, H. Tan, W. Luo and H. Mao, "MR-DBSCAN: an efficient parallel density-based clustering algorithm using mapreduce", In 17th IEEE International Conference on Parallel and Distributed Systems, pp. 473-480, 2011.

[38] M. Patwary, D. Palsetia, A. Agrawal, W. K. Liane, F. Manne and A. Choduary, "A new scalable parallel DBSCAN algorithm using the disjoint-set data structure", In IEEE International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1-11, 2012.

[39] H. P. Kriegel and M. Pfeifle, "Hierarchical density-based clustering of uncertain data", In 5th IEEE International Conference on Data Mining, 2005.

[40] P. Goyal, S. Kumari, D. Kumar, S. Balasubramaniam, N. Goyal, S. Islam and J. S. Challa, "Parallelizing OPTICS for Commodity Clusters", In Proceedings of the ACM International Conference on Distributed Computing and Networking, pp. 33, 2015.

[41] S. K. Popat and M. Emmanuel, "Review and Comparative Study of Clustering Techniques", International Journal of Computer Science and Information Technologies, vol. 5, no. 1, pp. 805-812, 2014.

[42] X. Wang and Howard J. Hamilton, "A comparative study of two density-based spatial clustering algorithms for very large datasets", Advances in Artificial Intelligence, Springer, pp. 120-132, 2005.

[43] G. Karypis, E. H. Han and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling", Computer, vol. 32, no. 8, pp. 68-75, 1999.

[44] S. Guha, R. Rastogi and K. Shim, "CURE: an efficient clustering algorithm for large databases", In ACM SIGMOD Record, vol. 27, no. 2, pp. 73-84, 1998.

[45] R. T. Ng and J. Han, “Efficient and Effective Clustering Methods for Spatial Data Mining”, In proceedings of International Conference on Very Large Databases, 1994.

# List of Publications and Video Link

## Publications

- **Accepted**

R. Gulati and R. Rani, "*Study and Analysis of Density Based Clustering Algorithms for Spatial data*", In IEEE International Conference on Advanced Communication, Control and Computing Technologies (ICACCT), 2016.

## Video Link

https://youtu.be/llqXp-dV1FY

ORIGINALITY REPORT

| 6% | 2% | 5% | 0% |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | Nagpal, Arpita, Arnan Jatain, and Deepti Gaur. "Review based on data clustering algorithms", 2013 IEEE CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGIES, 2013.<br>Publication | <1% |
| 2 | www.cs.toronto.edu<br>Internet Source | <1% |
| 3 | Naijun Wu. "VDBSCAN: Varied Density Based Spatial Clustering of Applications with Noise", 2007 International Conference on Service Systems and Service Management, 06/2007<br>Publication | <1% |
| 4 | lectureonline.cl.msu.edu<br>Internet Source | <1% |
| 5 | dspace.thapar.edu:8080<br>Internet Source | <1% |
| 6 | Yuhen Hu. "Correlation-based document clustering using web logs", Proceedings of the 34th Annual Hawaii International Conference | <1% |