International Research Journal of Advanced Engineering and Science
ISSN (Online): 2455-9024

A Robust Density-Based Clustering Approach Using DBCURE –MapReduce Techniques

Dr. S. Hari Ganesh[1], K. Shanmugavadivu[2]
[1]Assistant Professor, H. H. The Rajah's College, Pudukkottai- 622001
[2]Research Scholar, H. H. The Rajah's College, Pudukkottai- 622001
Email address: [1]hariganesh17[AT]gmail.com, [2]shanmugaa2786[AT]gmail.com

***Abstract*— *Clustering is the process of grouping similar data into clusters and dissimilar data into different clusters. Density-based clustering is a useful clustering approach such as DBSCAN and OPTICS. The increasing volume of data and varying size of data sets lead the clustering process challenging. So that we propose a parallel framework of clustering with advanced approach called MapReduce. We develop a new algorithm DBCURE which is modified from DBSCAN. Next we propose a newly developed parallel algorithm called DBCURE-MR that is DBCURE algorithm is parallelized with MapReduce framework. This proposed method finds clusters in parallel manner efficiently.*

***Keywords*— *MapReduce, DBCURE, density based clustering, parallelization algorithm.*

I. INTRODUCTION

The growing trend of scientific applications is being expected to deal with big data. [6] For such data concentrated applications, the MapReduce structure in recent times has engrossed a lot of concern. MapReduce is a programming model that allows uncomplicated development of scalable parallel applications to process big data on bulky clusters of service machines. MapReduce or its open-source counterpart Hadoop is a influential tool for building such applications. In recent times several data mining algorithms have been parallelized with MapReduce.

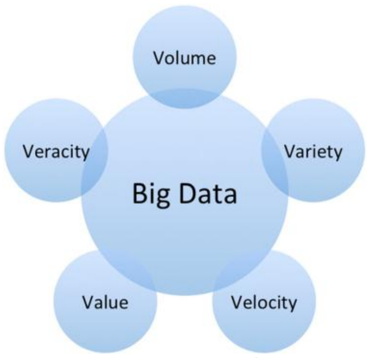


Fig. 1. Five dimensions of big data.

Data clustering is an important data mining technology that plays a crucial role in copious scientific applications. Clustering is a process of aggregating data points such that the points within a single collection have similar characteristics, whereas the points in different collection are disparate. It plays a vital role in numerous applications such as pattern recognition, information retrieval, social networks, image processing, molecular biology, medical imaging and multimedia. There are a variety of clustering algorithms such as clustering in partitional basis, in hierarchical basis and based on density and fuzzy clustering etc. These clustering algorithms are classified in accordance with the creation of clusters of objects. Determining good clusters is one of the main goals of clustering algorithms.

DBSCAN is an valuable density-based clustering method which was first proposed in 1996. While comparing with other clustering algorithms, DBSCAN holds several attractive chattels. First, it divides data into clusters with random shapes. For example, it can find clusters entirely bounded by another cluster. Second, DBSCAN will not consider about the number of the clusters as a priori. Third, it is insensate to the order of the points in the dataset. As a result, DBSCAN has achieved great success. Conversely, DBSCAN performing efficiently in real-world applications is challenging as a consequence of two reasons. First, the sizes of the datasets are growing promptly so that they cannot be held on a single machine to any further extent and varying shapes of the datasets cannot be handled by DBSCAN. Second, DBSCAN come at a much higher computation complexity compared with other clustering methods. In this paper we address these issues by proposing a serial density based clustering algorithm namely DBCURE-Density Based ClUsteRing algorithm for largE data that handle the datasets of variable densities and from that we build up a proficient parallel algorithm DBCURE-MR using MapReduce.

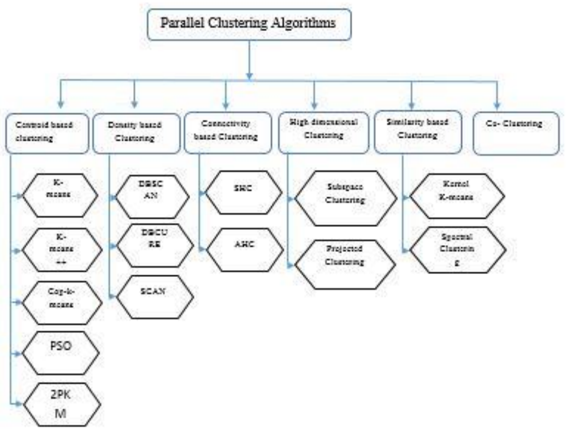


Fig. 2. Types of parallel clustering algorithms.

MapReduce is a popular parallel programming platform based on shared-nothing architecture. It received great accomplishment due to its simplicity, scalability and fault tolerance. Precisely MapReduce provides users with readily usable programming interfaces while hiding the disorganized details for parallelism. Moreover MapReduce rifts the jobs into small tasks and materialize the intermediate results close by. It can scales up with thousands of commodity nodes where node failure is normal. With experimental results we prove that DBCURE as well as DBCURE-MR discerns the clusters of varying densities well and DBCURE-MR steps up well with MapReduce framework.

II. LITERATURE REVIEW

DBSCAN [1] is a density based clustering that could produce arbitrary number of clusters in despite of the distribution of spatial data, whereas the K-means is a centroid based algorithm that possibly will find estimated clusters of a definite number. [8] In a density based clustering method on distributed systems was proposed by utilizing a distributed spatial index call dR* tree and thus it is difficult to be adopted in the MapReduce framework. To alleviate the sensitivity to density parameters OPTICS was proposed in [9], and it performs clustering with changing the radius Ɛ to find clusters with varying densities. However it cannot be parallelized easily using MapReduce since the points should be merged serially to determine where to separate the clusters. [1] MapReduce is a programming paradigm for data intensive applications. Due to its simplicity, MapReduce can effectively handle failures and thereby can be scaled to thousands of nodes. [1, 10] Parallelization of DBSCAN using MapReduce was proposed in [10].

III. RESULTS AND DISCUSSIONS

In this section we focus on finding the efficient parallel clustering algorithm on big data having varying densities.

MapReduce Overview

MapReduce Programming Model MapReduce is a software framework that is a basis computational model of current cloud computing platform. Its key task is to handle substantial amount of data. Because of its simplicity, MapReduce can effectually deal with machine failures and certainly expand the number of system nodes.

In the MapReduce programming model [2] data is perceived as a series of key-value pairs like, the sequential process of MapReduce subsists of three stages: Map, Shuffle, and Reduce. User only writes map and reduce functions. In the Map phase, a map task corresponds to a node in the cluster, by means of the other word, multiple map tasks are be running in parallel at the same time in a cluster. Every map call is given a key-value pair (key1, val1) and generates a list of (key2, val2) pairs. The output of the map calls is reassigned to the reduce nodes (shuffle phase). All the intermediate records with the same intermediate key (key2) are sent to the same reducer node. At each reduce node, the received intermediate records are sorted and clustered (all the midway records with the same key form a single group). Each group is processed in a single reduce call. The data processing can be summarized as follows:

$$\text{Map(key1,val1)} \longrightarrow \text{list(key2, val2)}$$
$$\text{Reduce(key2,list(val2))} \longrightarrow \text{list(key3, val3)}$$

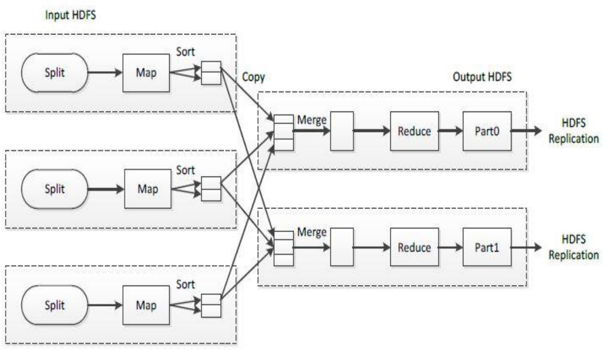


Fig. 3. Workflow of MapReduce.

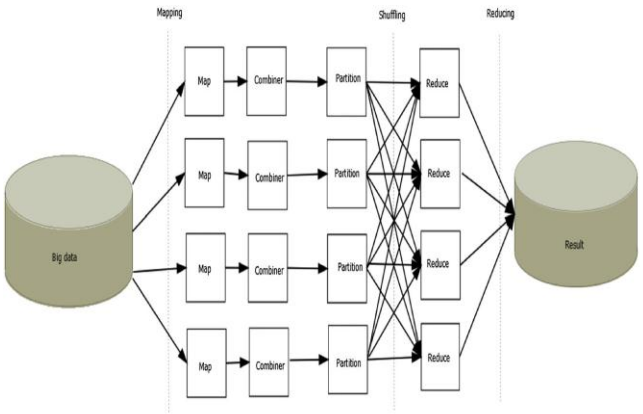


Fig. 4. Architecture of MapReduce.

DBCURE: A Clustering Algorithm

Alike DBSCAN algorithm our DBCURE also recurrently implements two step procedure. In Step 1, it chooses a kernel that is an arbitrary core point, which is an unvisited point in dataset D. Then it insert the kernel or core point into the set S which is collection of unvisited core points. Next in step 2, it finds all the data points which are density reachable to the core points in set S. Then it takes a point from set S and adds its τ-neighborhood into set S. By repeatedly adding this neighborhood of every core point p € S into S until S becomes empty we can form a cluster. From that we can prove the discovered clustering result obtained by DBCURE satisfies the definition of density based clustering by a proof.

Definition: Density-reachable

"A point p is density-reachable from a point q with respect to Eps and MinPts if there is a chain of points b1…,bn, b1=q, bn=p such that bi+1 is directly density reachable from bi." [1] Figure 5 [1] shows an illustration of a density-reachable point

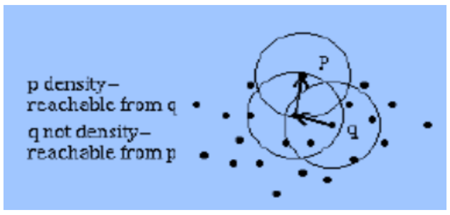


Fig. 5. Point p is density-reachable from point q and not vice versa.

DBCURE-MR: A parallel density based clustering algorithm using MapReduce

The contour of DBCURE-MR is having the following steps:
Step1: Estimation of neighborhood covariance matrices: In this step we estimate the neighborhood co-variance matrix for every core point in set S.
Step2: Computation of ellipsoidal τ-neighborhoods: This step discovers all pairs of point, each of which is within both of their τ-neighborhoods, by performing similarity joins.
Step3: Discovery of core clusters: For a core point p, the set of p and its directly density-reachable points is called a core cluster. So we obtain the list of ellipsoidal τ-neighborhood points for each core point in dataset D.
Step4: Merging of core clusters: Finally the core clusters which are partaking the common core points are grouped repeatedly to generate the ultimate cluster

	Algorithm	Output
Step1	COVMATX1- $\downarrow$	Statistics to estimate Covariance matrix.
	COVMATX2-	Neighborhood covariance matrix for every point.
Step2	MAXMBR-MR $\downarrow$	Maximum width in the MBR of (∑,τ)-ellipses of all points
	τ-NEIGHBOR-MR	All pairs of points each of which includes the other one in its τ-neighborhood
Step3	FINDKERNAL-MR	Every core cluster
Step4	MERGE-CLUST-MR	Final clustering Result

Fig. 6. An overview of DBCURE-MR workflow.

Covariance Matrix: COVMATX-MR:

To compute covariance matrix rapidly we have to fragment the given cluster space into number of grids by segregating each extents of same length in T intervals. And then we compute neighborhood covariance matrix for each individual grids and the map function of COVMATX generate key-value pairs (a, x_i) for each points x_i in dataset Dsimultaneously using COVMATX-MR algorithm.

Computing Ellipsoidal τ-neighborhoods for every points: τ-NEIGHBOR-MR

Afterwards we develop a new MapReduce algorithm τ-NEIGHBOR-MR algorithm. That discovers core points with their τ-neighborhoods. We can perform similarity join to find τ-neighborhood in an effectual manner by manipulating the Ɛ-tree which is used in previous research work. Every single pair of points having the distance at most Ɛ, will be splitted into identical rectangular grid with the width of Ɛ. Then we can estimate the distances for every pairs of points with one grid and its adjacent grid cells. Yet to find an ellipsoidal τ-neighborhood for each points we cannot divide the grid space in a uniform with in every situation. Therefore we compute MBRs for $\sum_i$, which means the sum of τ-ellipses of all points and then we find the maximum width in each dimension of all MBRs to use it as width of grids in Ɛ-tree. At this time we can surely said that every ellipsoidal τ-neighborhood of a point is continually in its own grid cell or its neighboring grid cells.

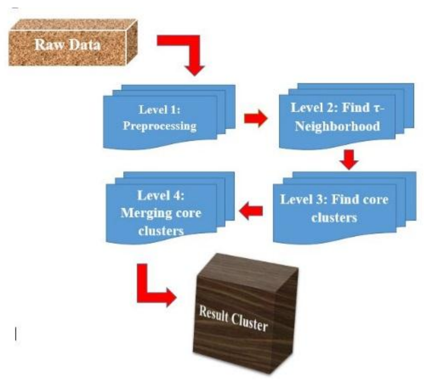

Raw Data
Level 1: Preprocessing
Level 2: Find τ-Neighborhood
Level 3: Find core clusters
Level 4: Merging core clusters
Result Cluster

Fig. 7. DBCURE-MR processing.

FINDKERNEL-MR: A MapReduce Algorithm to find a core point:

We next create an algorithm using MapReduce called FINDKERNEL-MR. That discovers the core clusters. As per definition of core point a point $x_i$is said to be core whose τ-neighborhood size is atleast δ and the output will be the point x_i and its τ-neighbor point. If it is not a core means it will give nothing as an output. The pair of core point and its neighbor point will form core cluster.

In this algorithm it takes the output of τ-NEIGHBOR-MR that means the neighbor point pairs (x_i, x_j) as an input and the output produced by map function in every stage is grouped in shuffling phase. And then a table will be produced with the list of point x_i and its τ-neighborhood produced in τ-NEIGHBOR-MR algorithm. In next phase namely reduce phase, reduce function is invoked for every pair of point and its τ-neighborhood (x_i, $N_\tau(x_i)$). When N_τ (x_i) +1 >= δ, x_i is said to be a core point and the reduce function outputs (x_i, N_τ (x_i)). When N_τ (x_i)+1< δ, x_i is said to be border point, so the reduce function outputs nothing. The result of FINDKERNEL-MR algorithm is the list of points and its τ-neighborhoods called as core cluster table.

MERGE-CLUST-MR: To produce resulting cluster by grouping every individual clusters:

This proposed work this algorithm merges clusters simultaneously with the help of MapReduce.
MERGE_CLUST-MR: We first conceptually split the points in M into multiple equi-sized partitions, M_1… M_K. We choose the number of partitions K to be at most the number of machines used for reduce functions. Let Si denote the set data structure whose nodes consist of the points in M_i only. The pointers to the parents in Si may point to the nodes in different Sjs (with j≠i). We decide the number of partitions K so that

every pair of Si and Sj together fit into the main memory of a single reduce function. We also partition the core cluster table R, output by FINDKERNEL-MR, into disjoint partitions, $R_1,\ldots,R_K$ such that every ⟨p, N_τ(p)⟩ in R_i satisfies p € Mi. In each R_i, whenever we see a core point x and its neighbor core point v with v € M_i, the clusters of two core points are merged in Si. However, when there is a core point x and its neighbor point v € M_j (i≠j), we output the key-value pair ⟨x, y⟩ to consider merging y to the cluster of x later, since we do not know whether v is a core point or not yet by looking at R_i only. We let P be such ⟨x, y⟩ s generated as output. MERGE-CLUST-PARTIAL performs the task of merging core clusters using MapReduce for every partition M_i in parallel.

After MERGE-CLUST-PARTIAL produces as output S_is and P, we iteratively select each Si as apivot and do the following by invoking MERGE-CLUST-ALL: For every pair of Si and S_j with j≠i, if there exists ⟨q, p⟩ in the output P, we merge the cluster of a point p in S_j to the cluster of a point q in Si if needed. Note that we merge the cluster of a point p in S_j to the cluster of a point q in Si, and not vice versa, whenever we need to merge them. However, if any ancestor node which does not belong to S_j is encountered while traversing the path from p to the root (i.e., representative) node, since we cannot update such ancestor nodes in the current execution of MERGE-CLUST-ALL, we output ⟨q, c_p⟩ where c_p is the first encountered ancestor node from p belonging to S_k (k≠j). Such output of all ⟨q, c_p⟩ s will be processed together as the post-processing step by MERGE-CLUST-FINAL in a single machine.

Given a core cluster table R, we first invoke MERGE-CLUST-PARTIAL in order to obtain K disjoint-set data structures $S_1,\ldots, S_K$ and the set of point pairs P, which consists of the pairs(x, y) such that x and y are from different partitions M_i and M_j(i≠j), and the clusters of x and y should be possibly combined in later steps. Then, for loop in executes MERGE-CLUST-ALLK times repeatedly. At the i-th iteration, Si is broadcast to every reduce function before calling MERGE-CLUST-ALL. Finally, MERGE-CLUST-FINAL is called.

MERGE-CLUST-PARTIAL: For each core cluster ⟨x, N_τ(x)⟩ in R where x € Mi, a map function is called and emits the key-value pair ⟨i, (x, N_τ(x))⟩, to partition the core cluster table R into disjoint partitions $R_1,\ldots,R_K$ such that every ⟨p, N_τ(p)⟩ in R_i satisfies p €M_i.

Afterwards the key-value pairs are congregated by keys in the shuffling phase of the MapReduce framework, fo reach distinct key i, a reduce function is called with Ri as input and merges the core clusters by updating the set data structure Si, which initially consists of single point clusters only. At the time of merging the core clusters in R_i, if a core point x € M_i and its neighbor point y € M_j with j≠i have to be unioned, since we do not know whether v is a core point or not in this reduce function with key i locally, we output the key-value pair ⟨x, y⟩ to consider later whether we merge v into the cluster of x or not. If MERGE-CLUST- PARTIAL is done, we write the data structure S_i on the distributed file system.

MERGE-CLUST-ALL: After MERGE-CLUST-PARTIAL produces S_is and P, MERGE-CLUST-MR iteratively selects each S_i as a pivot and invokes MERGE-CLUST-ALL to merge the clusters in every S_j (j≠i) to the clusters in S_i. At the i-th iteration of the for loop in MERGE-CLUST-MR, Si is broadcast to every reduce function before the execution of MERGE-CLUST-ALL. For each key-value pair ⟨x, y⟩ output by MERGE-CLUST-PARTIAL, a map function is called. If x€ Mi and y € Mj, the key-value pair ⟨j, (x, y)⟩ is emitted so that the reduce function invoked with key j can merge the cluster of y to that of x by updating S_i and S_j. Then, for each distinct key j, a reduce function is invoked, but it also receives two disjoint-set data structures Si and Sj where Si is broadcast by the main function of MERGE-CLUST-MR and Sj is read from the distributed file system. If y is a core point, we merge the cluster of y to that of x in Si by using the function UNION-TO(v, c_x, Sj) which sets the representative node of y's cluster in Sj to point to c_x which denotes the node representing x's cluster. If y is not visited yet, since y is a border point which should be merged into the cluster of x, not only the cluster of y is merged to that of x but also the status of y is set to BORDER. If MERGE-CLUST-ALL in each iteration is done, we write the data structure Si on the distributed file system. While there duce function with key j merges the cluster of x € Mi to that of y € M_j, If any ancestor node is encountered while traversing the path from x to the root (i.e., representative) node, since we cannot update such ancestor nodes in the current execution of MERGE-CLUST-ALL, we output ⟨y; c_x⟩ where c_x is the first encountered ancestor node belongs to S_k (k≠j) from x. Werefer to such output of all ⟨y, c_x⟩s in the i-th iteration of the for loop in MERGE-CLUST-MR as F_i and we let $F = F_1U\cdots UF_K$. All S_is and F will be processed together as the post processing step by MERGE-CLUST-FINAL.

MERGE-CLUST-FINAL: We next merge the clusters, which could not be combined by MERGE-CLUST-ALL, by using MERGE-CLUST-FINAL. We call the serial function MERGE-CLUST- FINAL. Note that all S_is and F produced by MERGE-CLUST-ALL are stored in the distributed file system. Each pair(x,y) in F implies that the cluster of y needs to be merged into the cluster represented by the node x.

We first perform set-union operations based on(x, y) in F to simplify the updates to all S_is. For example, assume that (x, y) and(y, z) exist in F. If there is a node pointing to z in a S_i, we have to update the node to point to x rather than y. Thus, before updating any S_i, we perform set-union operations based on (x, y) in F and generate a disjoint-set data structure S_f in whose nodes consist of the distinct points appearing in F. Then, while reading Si from disk one by one, for every node z in S_i, we consult S_f in on whether to update the node or not. Suppose a node z points to the node y. If y exists in S_f in and the root node of y is x in S_f in, MERGE-CLUST-FINAL updates the node z to point to x. Moreover, if the status of each node in S_i is still UNVISITED, we set its status to OUTLIER.

IV. Evaluation

We conduct all the experiments of this algorithm on a 25 node cluster, each node consists of Intel(R) Pentium (R) CPU

A1018@ 2.10 GHz and 2GB RAM. The operating system used in the nodes is Windows 7. All nodes are hosted in a single track. All algorithms were implemented using JavaC Compiler of version 1.5. And we used ApacheHadoop 2.6.0 framework for MapReduce implementation. For the comparison of our experiments we use several algorithms to evaluate the performance of those algorithms with different inputs and the results are presented as follows.

TABLE I. Summary of datasets.

Dataset	Points	Size	Percentage of all
Set 1	0.4 Billions	1.77 GB	76.8%
Set 2	0.89 Billions	2.09 GB	80%
Set 3	2.1 Billions	3.91 GB	91%
Set 4	1.7 Billions	1.42 GB	66.1%

The above table defines the different size of data sets used for evaluation of different algorithms and the performance of all algorithms while dealing with various data set structure is noted to prove that DBCURE-MR as best among them.

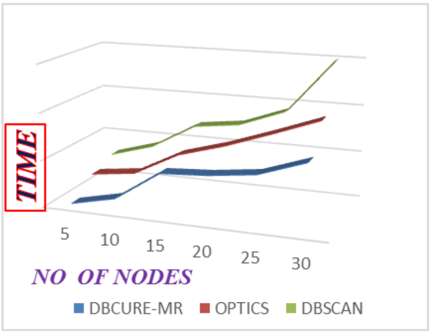
TABLE II. Average time taken on each phase.

Dataset	Map phase		Shuffle phase		Reduce phase	
	Load 1	Load 2	Load 1	Load 2	Load 1	Load 2
Set 1	16	27	102	413	350	158
Set 2	23	33	170	228	421	94
Set 3	43	10	120	324	399	157
Set 4	32	20	231	195	980	878

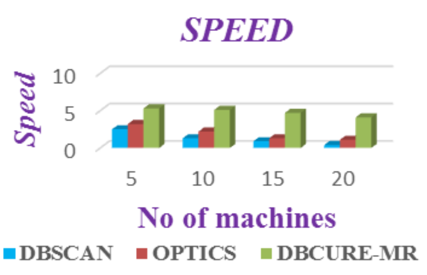
To reveal the speed of this algorithm we experiments the different workloads on different phases.

Comparison figures:

i) Experimentation on time in varying no. of nodes



ii) Experimentation on Speed:



Performance with real life data:

As per above figure our DBCURE-MR shows good speed up with real life datasets.

V. Conclusion

In this paper, we study the problem in density based clustering in parallelization with MapReduce framework. As we study about traditional DBSCAN which is difficult in parallelization of clustering varying density data sets. Next we tryout this problem by OPTICS algorithm which overcomes the weakness of DBSCAN. However OPTICS is hard to parallelize. Thus, we developed a new variant from DBSCAN called DBCURE, which have the advantage to work in parallel with MapReduce, Hadoop and can be work well with varying density datasets. We next develop the parallel version of DBCURE called DBCURE-MR which gives the correctness proof for efficiency and accuracy. By the experimental results, we showed that our DBCURE-MR finds the good clusters efficiently even it is large in volume and having different densities and scales up well with MapReduce framework.

References

[1] Y. He, H. Tan, W. Luo, H. Mao Di Ma, S. Feng, and J. Fan, "MR-DBSCAN: An efficient parallel density-based clustering algorithm using MapReduce," *IEEE 17th International Conference on Parallel and Distributed Systems*, 2011.

[2] R. Gulati and Dr. R. Rani, "Efficient parallel DBSCAN algorithms for bigdata using MapReduce," Computer Science and Engineering Department, Thappar university, 2016.

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

[4] S. Tabhane and Prof. R. A. Fadnavis "Large data computing using clustering algorithms based on Hadoop," *International Journal of Engineering Research and General Science*, vol. 3, issue 2, pp. 1056-1063, 2015.

[5] H. Backlund (henba892), A. Hedblom (andh893), N. Neijman (nikne866), "DBSCAN: A Density-Based Spatial Clustering of Application with Noise," Linköpings Universitet – ITN, 2011.

[6] Y. Kim, K. Shim, M.-S. Kim, J. S. Lee, "DBCURE-MR: An efficient density based clustering algorithm for large data using MapReduce," *Information Systems*, vol. 42, pp. 15-35, 2014.

[7] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, "MR- DBSCAN: a scalable MapReduce based DBSCAN algorithm for heavily skewed data," *Frontiers of Computer Science*, vol. 8, issue 1, pp. 83-99, 2014.

[8] X. Xu, J. Jäger, and H.-P. Kriegel "A fast parallel clustering algorithm for spatial databases," *Data Mining and Knowledge Discovery*, vol. 3, issue 3, pp. 263-290, 1999.

[9] M Ankerst, M. M. Breunig, H. P. Kriegel and J. Sander, "OPTICS: Ordering points to identify the clustering structure," *Proc. ACM SIGMOD'99 Int. Conf. on Management of Data*, Philadelphia PA, 1999.

[10] L. Li and Y. Xi "Research on clustering algorithm and its parallelization strategy," *International Conference on Computational and Information Sciences (ICCIS)*, 2011.

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density based algorithm for discovering clusters in large spatial databases with noise," *KDD'96 Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226-231, 1996.

[12] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Second Edition, 2006.

[13] K. Shim, R. Srikant, and R. Agarwal, "High dimensional similarity joins," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, issue 1, pp. 156-171, 2002.