Cludoop: An Efficient Distributed Density-based Clustering for Big Data using Hadoop

Yanwei Yu · Jindong Zhao · Xiaodong Wang · Qin Wang · Yonggang Zhang

Received: 24 October 2014 / Accepted: date

Abstract Density-based clustering for big data is critical for many modern applications ranging from Internet data processing to massive-scale moving object management. This paper proposes *Cludoop* algorithm - an efficient distributed density-based clustering for big data using Hadoop. First, we propose a serial clustering algorithm *CluC* by leveraging cell partition optimization and c-cluster to fast find clusters. *CluC* completes classification of the points using the relationships of connected cells around points instead of expensive completed neighbor query, which significantly reduce the number of distance calculation. Second, we propose the *Cludoop*, which can efficiently cluster very-large-scale data in parallel using already existing data partition on Map/Reduce platform. It employs the proposed serial clustering *CluC* as a plugged-in on parallel Mapper, along with a cell description instead of completed cell in transmission to reduce both network and I/O costs. Guided by proposed cell-based principles, we also design a Merging-Refinement-Merging 3-steps framework to merge c-clusters on the overlay of assigned pre-clustering result on Reducer. Finally, our comprehensive experimental evaluation on 10 network-connected commercial PCs, using both huge-volume real and synthetic data, demonstrates that (1) the effectiveness of our algorithm in finding

Y. Yu · J. Zhao
School of Computer and Control Engineering, Yantai University, Shandong, China, 264005.

E-mail: {yuyanwei,zjd}@ytu.edu.cn

X. Wang · Q. Wang
School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China,, 100083.
E-mail: wangqin@ies.ustb.edu.cn

Y. Zhang
Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University
E-mail: zhangyg@jlu.edu.cn

correct clusters with arbitrary shape, and (2) Our proposed algorithm exhibits better scalability and efficiency than state-of-the-art method.

Keywords Distributed data mining · clustering · density-based · Hadoop · big data

1 Introduction

Clustering is to group data objects into different classes or clusters, and the objects within a cluster have high similarity, while objects in inter-cluster differ significantly with one another. Clustering has played a crucial role in numerous applications ranging from pattern recognition, mobile sensor networks, moving object management to location-based service. With the rise of big data science, clustering analysis has attracted a considerable interests in the big data mining, while clustering based on density is very useful to distance-based data mining in many applications with increasingly large-scale data owing to their capability to discover clusters with arbitrary shape.

Existing density-based algorithms such as DBSCAN [1], OPTICS [2], DENCLUE [3] and GDDSCAN [5] can obtain better groups of data points in the static large-scale and high-dimension databases, which were widely applied into many applications in the past decade. However applying the algorithms into current data-intensive applications is challenging due to rapidly increasing distributed stored big data. For example, the traffic data (GPS trajectories and infrared acquisition) from intelligent transportation system in Jiangsu Province reaches at 6.94 billions records, and 4 TB sensor data were collected from older's healthcare monitoring in Shanghai one week, the emerging wearable devices will further promote coming of big medical data era. The data set released by Twitter is large than 133 TB, increased data updated by Tencent's mobile applications reaches about 200 to 300 TB one day, and even the operational data of Yahoo reaches at 5 PB. When the amount of data is this large, it is impossible to handle the data using the serial clustering methods on a single machine. Therefore, the best clustering algorithm is the one that combines (a) a scalable effective serial algorithm, (b) makes it run efficiently in distributed platform and (c) don't need to preprocess all dataset.

For big data analysis, Map/Reduce and its open source equivalent Hadoop have attracted a lot of attentions due to its parallel way to handle massive-scale data. Hadoop is a desirable distributed computing platforms based on a shared-nothing cluster-architecture. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Hadoop adopts HDFS (Hadoop Distributed File System) storage structure and simple Map/Reduce programming model. HDFS provides high-throughput access to application data and initially partitions data in multiple nodes, and data is represented as <key, value> pairs. In Map and Reduce phases, map and reduce functions take a key-value pair as input and then output key-value pairs. The map function is first called on different partitions of input data on each slave node in parallel. The outputs by Mappers are next grouped and merged by

each distinct key. Then a reduce function is invoked for each distinct key with the list of all values sharing the key. Finally The Map/Reduce framework executes the main function on a single master machine to postprocess the outputs of reduce functions. Depending on the applications, a pair of map and reduce functions can be executed once or nested multiple times. Ever since Hadoop was first introduced in 2003, Map/Reduce received great success due to it allows easy development of scalable parallel application to process big data on thousands of commodity nodes, tolerating the failure nodes in the process. Something also already reflected in academia (see [4] for examples). However, finding density-based clusters from big data is a very challenging problem today.

This paper focuses on the problem of efficient, effective and scalable density-based clustering for big data. Our method employs Hadoop as computing platform, and incorporates cell partition and c-cluster optimizations into density-based clustering. Our major research work are (a) how to minimize the communication (network) cost among processing nodes, (b) how to avoid to preprocess the data, namely reducing the I/O cost, and (c) how to extract exact density-based clusters. So this paper proposes the distributed density-based **Clu**stering using Ha**doop** - *Cludoop* method, that efficiently handles large-scale data without any preprocess.

The main contributions of this paper include: 1) We propose a c-cluster definition along with cell-connected observations to significantly reduce computational cost of neighbor range query. 2) We propose an efficient serial clustering algorithm *CluC* leveraging c-clusters and neighbor searching optimization. 3) We propose the distributed clustering framework for big data using Map/Reduce structure based on the proposed serial clustering algorithm as a plugged-in on Map, along with a 3-steps merging framework on Reduce. 4) We conduct comprehensive experiments on Hadoop platform deployed on 10 commodity machines to evaluate the performance of Cludoop using larger-scale real and synthetic data. The results show that our method are both effective and efficient.

The paper is organized as follows: we first discuss related work in section 2. In section 3, we present preliminary notions and problem statement. Section 4 introduces theoretical ideas and presents the serial clustering algorithm. In section 5, we then present the distributed Cludoop algorithm. In section 6, we perform an experimental evaluation of the effectiveness, efficiency and scalability of our algorithm. Section 7 concludes whole paper with a summary.

2 Related Work

In this section, we mainly review related work in the areas of density-based clustering, parallel variants of DBSCAN, and distributed clustering on Map/Reduce platform.

Density-based clustering. Density-based methods describe the clusters being the high density area of points separated from the low-density region-

s in the data space, so clusters exhibit arbitrary shapes in the high-density region. There are two types of density-based clustering method: algorithms based on local connectivity such as DBSCAN [1] and OPTICS [2], and algorithms based on density function such as DENCLUE [3]. DBSCAN determines a non-hierarchical, disjoint partitioning of the data into several clusters. Clusters are expanded starting at arbitrary seed points within dense areas. Objects in areas of low density are assigned to a separate noise partition. DBSCAN is robust against noise and the user does not need to specify the number of clusters in advance. OPTICS is proposed to solve parameters selection problem on density-based clustering algorithms. The paper propose two concepts: core-distance and reachability-distance to organize points. The point objects would be ordered by reachability-distance and their core point to obtain the clustering structure, which contains a hierarchical clusters under a broad range of parameter settings. OPTICS visualizes clearly the cluster structure via the ordered point list, and could find arbitrary shaped clusters and overlapping clusters. DENCLUE formalizes the cluster notion by nonparametric kernel density estimation based on modelling the local influence of each data object on the feature space by a simple kernel function, e.g. Gaussian. It defines a cluster as a local maximum of the probability density.

Parallel variants of density-based clustering. Xu et al [6] propose a parallel clustering algorithm PDBSCAN for mining large distributed spatial databases. It uses the so-called shared nothing cluster architecture, which has the main advantage that it can be scaled up to a high number of computers. However it is not fully paralleled while it still needs a single node to aggregate intermediate results. Januzaj et al [7] depict a distributed clustering based on DBSCAN, DBDC, formed local and global two-level clustering. The local clustering are carried out independently on local data, then global clustering are done on a central server based on the transmitted representatives from local clustering. Subsequently, they design a density-based distributed clustering [8] which allows a user-defined trade-off between clustering quality and the number of transmitted objects from the local sites to the global server site based on DBDC idea. They fist order all objects located at a local site according to a quality criterion reflecting their suitability to serve as local representatives, and then send the best of these representatives to a server site where they are clustered with a slightly enhanced DBSCAN algorithm to obtain high quality clusters. Dash et al [9] propose a parallel hierarchical agglomerative clustering based on partially overlapping partitioning on shared memory multiprocessor architecture for handling nested data. The experiment shows the algorithm achieves near linear speedup. Brecheisen et al [10] present a parallel DBSCAN on a workstation, which is parallelized by a conservative approximation of complex distance functions, based on the concept of filter merge points. The final result is derived from a global cluster connectivity graph. Böhm et al [11] implement several data mining tasks under the highly parallel environment of GPUs, including similarity search and clustering. For density-based clustering, they design a parallel DBSCAN algorithm supported by their proposed similarity join on GPU. They then propose a massively parallel density-based

clustering method [12] using GPUs by leveraging the high parallelism combined with a high bandwidth in memory transfer at low cost. Andrade et al [13] also propose a GPU parallel version of DBSCAN, named G-DBSCAN, using graph to index point objects with less than a given distance threshold to each other. However the parallel density-based clustering can not straightforward be transferred on the Hadoop, because GPU-capable parallel algorithms do not only have shared main memory but groups of the processors even share very fast memory units, while Hadoop uses a distributed file system.

Distributed clustering on Map/Reduce. Ene et al [14] develop partition-based clustering algorithms, k-center and k-median, running on Map/Reduce, which use sampling to decrease the data size and run in a time constant number of Map/Reduce rounds. For density-based clustering, Yaobin He et al [15] propose a parallel DBSCAN using Map/Reduce framework, MR-DBSCAN, which partitions all spatial data into different Maps by the space location in the preprocessing stage, and then performs DBSCAN in each mapper and merging the bordering spaces in Reduce step. Similarly, Biru Dai et al [16] also propose a Map/Reduce-based DBSCAN with a data partition, partition with reduce boundary points (PRBP), selecting partition boundaries based the distribution of data points. However the data partition based on data space easily causes load unbalancing due to the sparse data. Subsequently, Yaobin He et al [17] also propose a load balancing mechanism based on a cost-based spatial partitioning for heavily skewed data. However above methods expend I/O cost due to preprocessing, especially for the distributed stored big data. This is also one of key points which we resolve in this paper. Cordeiro et al [18] present a clustering solution to find subspace for high-dimensional data using Map/Reduce, however they focus on the tradeoff problem between the I/O cost and network cost, and aims to dynamically choose the best strategy. These techniques didn't explore the optimization opportunities enabled by the nature insights of serial density-based clusters, this is exactly what our work does for delivering highly scalable distributed solutions along with an efficient optimized serial density-based clustering.

3 Preliminaries and Problem Statement

3.1 Definitions and Theorems

This section introduces the definitions of related terms based on the notion of connected density in [1]. Eps and $MinPts$ are the neighbor range parameter and the minimum number of neighbor points, respectively.

Definition 1 (Eps-Range) The Eps-range of a point p is the circular area with radius Eps centered on the specific point p.

Definition 2 (Eps-Neighbor) All points included in Eps-range of the point p is called Eps-neighbor of p, denoted by $N_{Eps}(p)$.

Therefore, $N_{Eps}(p)$ is a set of data points. Let $|N_{Eps}(p)|$ denote the cardinality of $N_{Eps}(p)$. Obviously, the neighbor is symmetric for pairs of points.

Definition 3 (Core point) A point p is called a core point if $|N_{Eps}(p)| \geq MinPts$.

Definition 4 (Border point) For a point p, if $|N_{Eps}(p)| < MinPts$ and $\exists q(q \in N_{Eps}(p))$ is a core point, the point p is a border point of the cluster including point q.

Definition 5 (Isolated point) A point p is classified as an isolated point if it is neither a core point nor a border point.

Note that isolated points can be considered as either anomalous points or noise.

Definition 6 (Directly density reachable) A point p is directly density reachable from another point q, if the point p is one of Eps-Neighbor of q, and q is a core point.

The Eps-Neighbor of a core point p are directly density-reachable form p, and the border points are directly density-reachable from their Eps-Neighbors which are core points.

Definition 7 (Density reachable) A point p is density reachable from a point q, if there is a chain of points $p_1,\ p_2, \ldots p_n,\ p_1 \leftarrow p, p_n \leftarrow q$ such that p_{i-1} is directly density reachable form p_i.

Definition 8 (Density connected) A point p is density connected to a point q, if there exists a point o such that both p and q are density reachable from o.

Definition 9 (Density-based cluster. Abbr. d-cluster) Let D is points set. A density-based cluster is a non-empty subset of D including at least a core point and all points which are density reachable from the core point.

Note that all core points would be classified in density-based clusters.

Lemma 1 *If a core point p belongs to two d-clusters C_1 and C_2 respectively, the C_1 and C_2 can be merged into one d-cluster.*

Proof: Since point $p \in C_1$ and p is a core point, then $N_{Eps}(p) \in C_1$. According to the *Def. 9*, the d-cluster C_1 can be expanded to $C_1 = \{o|o \in D$ and o is density reachable from $p\}$. Similarly, due to $p \in C_1$, C_2 also be equivalent to $\{o|o \in D$ and o is density reachable from $p\}$. So C_1 and C_2 should be merged into one d-cluster. In summary, a core point only belongs to a d-cluster.

Fig. 1 shows an example demonstrating Lemma 1, p is a core point, and belongs both of blue and green d-clusters, so the two d-clusters would be merged.

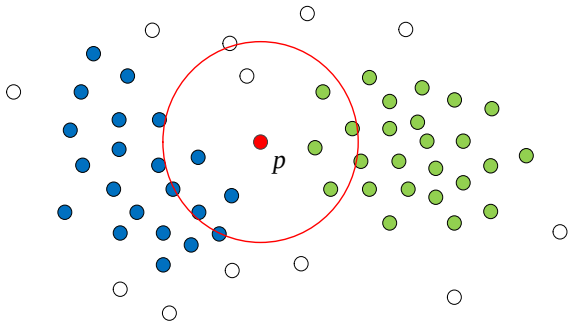


Fig. 1 An example of Lemma 1: p is a core point

Definition 10 (Cell) According to the Eps, the data space is divided into cubes whose length of the sides is $Eps/(2 * \sqrt{2})$, and each partition of space is called a $Cell$. A cell is denoted by $G_{(d_1,d_2)}(d_1 = 2 * \sqrt{2} * x/Eps,\ d_2 = 2 * \sqrt{2} * y/Eps)$, if the cell's center is $(x,\ y)$ in 2D space.

Let $|G_{(d_1,d_2)}|$ denote the number of points falling in $G_{(d_1,d_2)}$. In this paper, we describe the concept of cell in 2D space for generality, though m-dimensional space could equally be applied.

Lemma 2 *Given a point p in the cell $G_{(x,y)}$, if $\sum_{i=-1,j=-1}^{1,1} |G_{(x+i,y+j)}| \geq MinPts$, then point p is a core point.*

Proof: Lemma 2 can be easily proven using the Definition of $Cell$. Since the edge length of cell is $Eps/(2 * \sqrt{2})$, the distance between any point in the cells $G_{(x+i,y+j)}(i = -1, 0, 1; j = -1, 0, 1)$ and p is not greater than Eps. Therefore, if $\sum_{i=-1,j=-1}^{1,1} |G_{(x+i,y+j)}| \geq MinPts$, then $N_{Eps}(p)$ must be $\geq MinPts$, thus p is a core point.

Thus all points in the cells $G_{(x+i,y+j)}(i = -1, 0, 1; j = -1, 0, 1)$ are directly density reachable from p. These 9 cells are called *Inner Cells* w.r.t. p. By the Lemma 2, we can further deduce Corollary 1.

Corollary 1 *Given a cell $G_{(x,y)}$, if $\sum_{i=-1,j=-1}^{1,1} |G_{(x+i,y+j)}| \geq MinPts$, then $\forall p(p \in G_{(x,y)})$ is a core point.*

Lemma 3 *Given a point p in the cell $G_{(x,y)}$, there exist at most 36 cells in which the points need to check when verify whether p is a core point or not.*

Proof: Lemma 3 is intuitive by the Lemma 2 and Definition of Eps-Neighbor (Def. 2). We directly demonstrate this in the Fig. 2. Supposed the central cell is $G_{(x,y)}$. First, for $\forall p(p \in G_{(x,y)})$, the Eps-Range of p must be located within the area according to the side of cell. Second, all of the 36 cells would be covered in the Eps-Range of some point in $G_{(x,y)}$, in extreme cases, the No. 17 to No. 36 cells are covered when the p is located at four vertex. These 36 cells are called *Outer Cells* w.r.t. p.

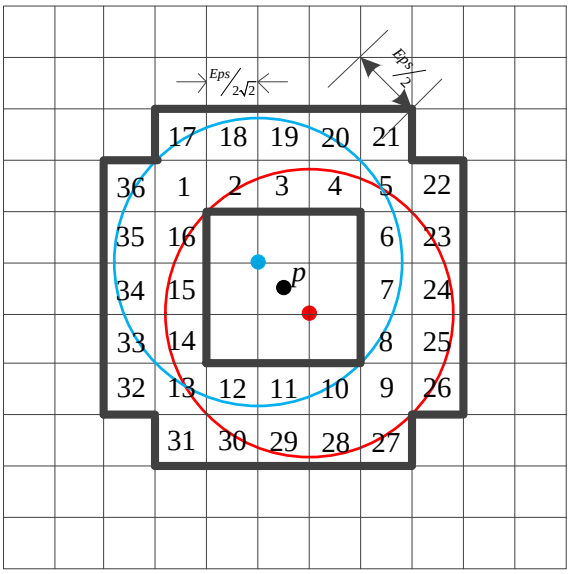


Fig. 2 Demonstration of Lemma 3

According to the d-cluster definition (Def. 9) and Lemma 2, we observe that if a core point p in the cell $G_{(x,y)}$ belongs to a d-cluster C, then all points in inner cells w.r.t. p belong to C. If all points falling in a cell G can be determined to belong to a d-cluster C, we call it an *Inclusive Cell* of C. Therefore, the d-cluster C can be represented by all inclusive cells and other border points not in the inclusive cells. Here we give a formalized definition of cell-cluster (abbr. c-cluster) by the cell-connected observation.

Definition 11 (Cell-cluster. Abbr. c-cluster) Given a d-cluster C, a cell-cluster is 2-tuple $\{IGs, Ps\}$, where IGs is the set of all inclusive cells of C, and Ps is the set of all border points not in the inclusive cells. The cell-cluster $\{IGs, Ps\}$ is called equivalent cell-cluster w.r.t d-cluster C.

C-cluster simplifies the d-cluster by leveraging inclusive cells instead of large amounts of density-connected points. We no longer check the status of every point in the clustering, instead only change the cluster identifier of the inclusive cells. Therefore, we use c-cluster instead of d-cluster in the subsequent clustering process.

Definition 12 (Exclusive Cell) Given a cell $G_{(x,y)}$ and a c-cluster C, if there exists a core point in both of $G_{(x,y)}$ and C, then $G_{(x,y)}$ is called an exclusive cell of c-cluster C.

Obviously, an exclusive cell must be an inclusive cell of the c-cluster, while an inclusive cell may not be an exclusive cell.

Lemma 4 *Given a cell $G_{(x,y)}$ and a c-cluster C, if $G_{(x,y)}$ is an exclusive cell of C, then the inner cells of $G_{(x,y)}$ must be inclusive cells of c-cluster C.*

Proof: Lemma 4 is intuitive. Since there exists one core point p in $G_{(x,y)}$ belongs to the d-cluster corresponding to C, the all points in inner cell of p belong to the d-cluster, thus the inner cells are inclusive cells of c-cluster C.

Lemma 5 *Given a cell $G_{(x,y)}$ and a c-cluster C, if $G_{(x,y)}$ is an exclusive cell of C, then any c-clusters containing $G_{(x,y)}$ as an inclusive cell can be merged into C.*

Proof: Lemma 5 is similar with Lemma 1. Since $G_{(x,y)}$ is an exclusive cell of C, there exists at least one core point p in $G_{(x,y)}$ belongs to the d-cluster corresponding to C. If there exist a c-cluster C' that $G_{(x,y)} \in C'.IGs$, then p also belongs to the counterpart of C'. Thus C' should be merged into C by the Lemma 1.

3.2 Problem Statement

In this work, we focus on the efficient parallel solution of finding density-based clusters from big data on Hadoop platform. Next, we define the distributed clustering problem based on the above definitions, which this paper just aims to resolve.

Problem : **(Distributed clustering with c-cluster)** Given a massive-scale dataset D, parameter setting Eps and $MinPts$, and network-connected computers CP deployed Hadoop platform, the distributed clustering is that, we output exact c-clusters from D w.r.t the given Eps and $MinPts$ on the CP.

4 Proposed Serial Clustering with C-Cluster

Our proposed distributed clustering solution is designed based on one optimized serial clustering method that we propose next: clustering with c-cluster – $CluC$. CluC aims to find out c-clusters from a large-scale dataset. In the following, we present a basic version of CluC omitting details of data structure and additional information for understanding, as shown in Algorithm 1.

Eps and $MinPts$ are density parameters as same in DBSCAN. We can get the set of cells according to the dataset D and Eps. For each point p, if its cell and itself are not yet included in any c-cluster, the function of $ExpandCluster$ would be performed to get the c-cluster including all points density-connected to p. The $innerCells(G)$ stands for the set of 9 inner cells around G, including itself. CluC verifies whether p is a core point by the Lemma 2 and Lemma 3, no longer uses $N_{Eps}(p)$ via the completed neighbor range query.

The $C.IGs$ and $C.EGs$ maintain the set of inclusive cells and exclusive cells of c-cluster C respectively. The set $Gseeds$ keeps the unchecked inclusive cells, and $Gseeds.getFirst()$ returns the first cell in the set. CluC first expands the c-cluster by the connected cells by Lemma 4, as shown on lines

19-29, then further checks the border points in bordering cells as lines 40-48. $C.merge(getCluster(G.cid))$ merges the previous c-cluster of the cell G into the c-cluster C, this operator is performed by Lemma 5 because the cell G is an exclusive cell of C. The similar reason can be explained for $C.merge(getCluster(p.cid))$. However, the point labelled with $Noise$ on line 51 may be changed later, if it is density-reachable from some points.

Algorithm 1 Clustering with C-Cluster (CluC) Method

Input: dataset D, parameters: Eps and $MinPts$.
Output: c-clusters

```
 1: //D is Unclassified
 2: Get SetOfCells from D;
 3: //SetOfCells is Unclassified
 4: for each point p ∈ D do
 5:     Get Cell G ← p.coordinate;
 6:     if G.cid = Unclassified then
 7:         if p.cid = Unclassified then
 8:             Create c-cluster C ← Cluster(cid);
 9:             if ExpandCluster(p, G, D, C, Eps, MinPts) then
10:                 Create c-cluster C ← Cluster(cid++);
11:             end if
12:         end if
13:     end if
14: end for
15: function ExpandCluster(p, G, D, C, Eps, MinPts)
16:     if Σ|innerCells(G)| ≥ MinPts || p is a core point then
17:         C.IGs.add(G); C.EGs.add(G); Gseeds.add(innerCells(G));
18:         Gseeds.delete(G);
19:         while Gseeds <> Empty do
20:             G ← Gseeds.getFirst();
21:             if Σ|innerCells(G)| ≥ MinPts then
22:                 if G.cid <> Unclassified and G.cid <> C.cid then
23:                     C.merge(getCluster(G.cid)) // Merge two c-clusters;
24:                 end if
25:                 C.IGs.add(G); C.EGs.add(G);
26:                 Gseeds.add(innerCells(G));
27:             else if ∃p ∈ G, p is a core point then
28:                 if G.cid <> Unclassified and G.cid <> C.cid then
29:                     C.merge(getCluster(G.cid)) // Merge two c-clusters;
30:                 else if p.cid <> Unclassified and p.cid <> C.cid then
31:                     C.merge(getCluster(p.cid)) // Merge two c-clusters;
32:                 end if
33:                 C.IGs.add(G); C.EGs.add(G);
34:                 Gseeds.add(innerCells(G));
35:             else
36:                 C.IGs.add(G);
37:             end if
38:             Gseeds.delete(G);
39:         end while
40:         for each cell G ∈ outerCells(C.EGs) do
41:             if G.cid <> C.cid then
42:                 for each point p ∈ G do
43:                     if dist(p, C.EGs)≤ Eps then
44:                         C.Ps.add(p); p.cid ← C.cid
45:                     end if
46:                 end for
47:             end if
48:         end for
49:         return true;
50:     else
51:         p.cid ← Noise; return false;
52:     end if
53: end function
```

The $C.Ps$ keeps all border points of c-cluster C not in inclusive cells. The $outerCells(C.EGs)$ denotes set of all outer cells w.r.t. each cell in $C.EGs$. The $dist(p, C.EGs)$ represents the smallest distance between p and any points in the $C.EGs$. Namely, for the point p not in inclusive cells, if there exists any point q in the $C.EGs$ such that dist$(p, q) \leq Eps$, the p is a border point w.r.t. the c-cluster C.

CluC first expands the inclusive cells recursively to reduce a large amount of distance calculations, then it can fast searches the border points in the pruned space that eliminate the inclusive cells. Therefore, the CluC achieves an efficient also effective improvement compared with the state-of-the-art serial methods.

5 Cludoop: Distributed Density-based Clustering on Hadoop

5.1 Cludoop Framework

The overall architecture of our Cludoop framework is depicted in Fig. 3. Cludoop uses the existing data partition and does the parallel CluC algorithm in mappers, and then does the merging work based on intermediate c-clusters in reducers. The final c-clusters that consist of inclusive cell descriptions and a tiny amount of border points is obtained on one reducer/machine.

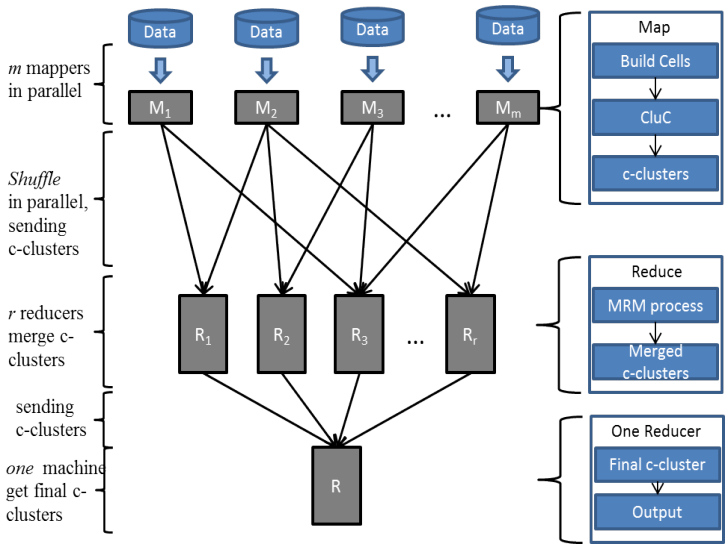


Fig. 3 Overall architecture of Cludoop clustering

5.2 Pre-clustering on Mapper

As shown in Fig. 3, Cludoop starts with m mappers reading the data in parallel from the HDFS, and employs CluC as a plugged-in clustering on each mapper. We call the phase pre-clustering to distinguish whole clustering. In this phase, we first build the cell index according to the received dataset, mapping the points into the cells. Then each mapper performs CluC algorithm on the cell-structure data in parallel, aiming to output the c-clusters and noises as pre-clustering result. However, the pre-clustering results need to be normalised and shipped to the appropriate reducers over the network, to get final clusters. Thus the network traffic and normalization would cost much CPU resources when ship all c-clusters and noises to reducers. How can we reduce the network cost for this large-scale dataset?

Our main idea is to only ship the c-clusters with inclusive cells' descriptions and the border points to reducers, rather than send all points which have already classified in c-clusters. Thus, we use a simple description ($the\ identifier\ of\ cell-ID$, $the\ number\ of\ points-n$, $is\ an\ exclusive\ cell\ ?$, $the\ identifier\ of\ c\text{-}cluster-cid$) to represent an inclusive cell, however the description provides sufficient information with reducers for further merging process. This is because (1) we build the cells according to the received points and given Eps in each mappers, so the cell with same ID in deferent mappers stands for the same area, thus we can only use the cell ID to locate the assigned range in overall data space. (2) the points in an inclusive cell must belong to a c-cluster, so the cell ID and c-cluster cid are enough to stand for classification of all points in itself. (3) The border points or noise in/nearby an inclusive cell in other mappers need to refine their status in reduce phase, however the number of points in the inclusive cell is sufficient to determinate their status. (4) By Lemma 5, the marked exclusive cell can be used to directly merge the c-clusters sharing the cell in the reduce phase. Using the inclusive cell's description, only the border points of c-cluster and noise need to be shuffled through the network to other machines. This significantly reduces the amount of data points shipped in the shuffling step, with the consequent savings for the network cost, as well as the I/O cost for the intermediate clustering results. Therefore, the c-cluster in subsequent reduce phase actually is the pruned c-cluster, namely only includes the inclusive cells' descriptions not the completed cells.

To efficiently merge the c-clusters located in near area on different mappers, meanwhile to avoid the computation unbalancing for skewed data, we combine the spatial distribution and uniform division in shuffle mechanism. We set r Keys according to the number of reducers, similar with spatial partitioning, we first normalize the mean coordinate of inclusive cells of each c-cluster to the corresponding nearest Key value. Thus the c-clusters with same Key value on different mappers would be send to same reducer in the shuffle step. Then we shift the c-clusters from crowded Key to near unoccupied Key for balancing the workload in reduce phase, So our Cludoop achieves a full load balancing throughout whole Map/Reduce phases.

5.3 Merging-Refinement-Merging framework on Reducer

In Reduce phase, each reducer receives the normalized c-clusters and noises, denoted $\{Cs,\ Ns\}$, where Cs is the set of c-clusters with cell's descriptions, and Ns is the set of assigned noise. We propose a Merging-Refinement-Merging 3-steps framework based on following cell-based merging and refinement principles on reducer to merge the assigned intermediate c-clusters.

First, we observe that the c-clusters should be merged when they share certain cells or connected cells. To characterize the merging process we formalize two merging rules.

Merging 1 *Given an exclusive cell $G_{(x,y)}$ in a c-cluster C, all c-clusters in Cs, which have $G_{(x,y)}$ as an inclusive cell or have any border point falling in $G_{(x,y)}$, can be merged into C.*

Merging 1 is intuitive. First, since $G_{(x,y)}$ is an exclusive cell of C, there must exist at least one core point in $G_{(x,y)}$. The core point also must be a core point of any c-clusters covering $G_{(x,y)}$ on the overlay of $\{Cs,\ Ns\}$. The $G_{(x,y)}$ also is an exclusive cell of the c-clusters, thus the c-clusters can be merged by Lemma 5. Second, if one border point p of c-cluster C' fall in $G_{(x,y)}$ on the overlay, then p also belongs to C, and p also would be changed to be a core point by the definition of exclusive cell. Therefore, the C and C' should be merged by Lemma 1.

Merging 2 *Given an exclusive cell $G_{(x,y)}$ in a c-cluster C, all c-clusters in Cs, which have any exclusive cell in the $(innerCells(G_{(x,y)}) - G_{(x,y)})$, can be merged into C.*

Merging 2 implies that two c-clusters covering one of two connected exclusive cells respectively should be merged into one cluster. This is obvious, because for any core points in these two cells are neighbors, thus the density-reachable core points would be classified into one cluster. Therefore, they should be merged.

Actually, Merging 1 already contains the some cases of Merging 2. For example, Given two connected cells G_1 and G_2, G_1 is an exclusive cell of c-cluster C_1, and G_2 is an exclusive cell of c-cluster C_2. Obviously, G_2 must be an inclusive cell of C_1, and G_1 also is an inclusive cell of C_2. Therefore, the two c-clusters would be merged by the Merging 1. However, Merging 2 supplies the processing method for the special case of vacant cells, such as the cell G_2 is null in the c-cluster C_1 from one mapper, and the cells G_1 is also null in C_2 from another mapper in the above example. In this case, Merging 1 can not work, while the Merging 2 complementally handles the situation.

Next, we analyse the refinement process for the non-exclusive inclusive cells, border points and noises on the overlay of $\{Cs, Ns\}$ in one reducer, and propose three refinement rules for these three situations, respectively.

Refinement 1 *For an inclusive cell $G_{(x,y)}$ of a c-cluster C, if $|G_{(x,y)}| \geq MinPts$ or $\sum_{i=-1,j=-1}^{1,1} |G_{(x+i,y+j)}| \geq MinPts$ on the overlay of $\{Cs, Ns\}$, then $G_{(x,y)}$ can be changed to be an exclusive cell of C.*

Refinement 1 can be easily shown. On the overlay of $\{Cs,\ Ns\}$, some inclusive cells can be directly determined their exclusivity according to the updated number of points in each cell.

Refinement 2 *Given a c-cluster C, for a border point $p \in C.Ps$, if $|G(p)| \geq MinPts$, or $\sum |innerCells(p)| \geq MinPts$, or $N_{Eps}(p) \geq MinPts$ on the overlay of $\{Cs,\ Ns\}$, then $G(p)$ can be marked to be an exclusive cell of C.*

Similar with Refinement 1, Refinement 2 depicts the update case for the border points not in inclusive cells on the overlay of $\{Cs,\ Ns\}$. Note that we can not compute the exact number of neighbors for point p, because we use the cell's description instead of the points in the inclusive cell. In refinement phase, we employ an approximate method to get $N_{Eps}(p)$ when check if p is a core point. If all vertexes of a cell in $outCells(p)$ whose distance to p is no large than Eps, then the point in the cell are neighbors of p, else if there exist vertexes whose distance to p is large than Eps, we calculate the approximate rate β of area covered in the *Eps-Range* of p in the cell, and use the $\beta * n$ to estimate approximately the number of neighbor for p in the cell.

Refinement 3 *Given a noise $p \in Ns$, if $|G(p)| \geq MinPts$, or $\sum |innerCells(p)| \geq MinPts$, or $N_{Eps}(p) \geq MinPts$ on the overlay of $\{Cs,\ Ns\}$, then $G(p)$ can be marked to be an exclusive cell of a new c-cluster.*

Refinement 3 indicates the update process for noise on the overlay of assigned pre-clustering results after first-round merging.

Based on the above merging and refinement rules, we describe the Merging-Refinement-Merging framework to merge the pre-clustering in Reduce phase. The pseudo-code of the framework is shown in Algorithm 2. First, we execute directly the first-round merging using the proposed two merging rules (Lines 2-10). This step would merge the overlapping c-clusters and noise in $\{Cs, Ns\}$ to fast reduce the number of c-clusters and noise, and update the sharing cells' point number n at the same time. Next, the status of non-exclusive cells, unchecked border points and noise are further refined on the overlay of $\{Cs, Ns\}$ (Lines 12-40). As shown in Algorithm 2, Refinement 1 is employed first for the non-exclusive cells. $G^{\{Cs,Ns\}}$ denotes the updated cell G on the overlay of $\{Cs, Ns\}$. Then Refinement 2 and Refinement 3 are performed in turn for unchecked border points and noise. In process of Refinement 1, most border points and noise candidates falling in the cells also are processed, this greatly reduces the number of points that need to be handle in Refinement 2 and Refinement 3. This is also why we first perform Refinement 1. More specifically we also first utilize Lemma 2 to check whether a cell is an exclusive cell or a point is a core point to reduce a large amount of distance calculate for most cases. Finally, second-round merging is performed to re-examine the c-clusters on the updated cells and border points and get the final c-clusters over $\{Cs, Ns\}$.

In the phase, r reducers execute the Merging-Refinement-Merging process in parallel. However we still need to perform one reducer to return final clusters on one machine in final phase as shown in Fig. 3.

Algorithm 2 Merging-Refinement-Merging framework on Reducer

Input: assigned $\{Cs, Ns\}$, parameters: Eps and $MinPts$.
Output: c-clusters

```
1: //First-round Merging
2: for each c-cluster C ∈ Cs do
3:     for each cell G ∈ C.IGs do
4:         if G.isExclusive == true then
5:             Merge all c-clusters by Merging 1;
6:             //update the number of point in sharing inclusive cells and border points
7:             Merge all c-clusters by Merging 2;
8:         end if
9:     end for
10: end for
11: //Refinement
12: for each c-cluster C ∈ Cs do
13:     for each cell G ∈ C.IGs do
14:         if G.isExclusive == False then
15:             if |G^{Cs,Ns}| ≥ MinPts || Σ|innerCells(G)^{Cs,Ns}| ≥ MinPts then
16:                 G.isExclusive ← true; seeds.add(G);
17:                 C.IGs.add(innerCells(G));
18:             end if
19:         end if
20:     end for
21:     for each point p ∈ C.Ps do
22:         if |cell(p)^{Cs,Ns}| ≥ MinPts || Σ|innerCells(p)^{Cs,Ns}| ≥ MinPts || N_Eps(p) ≥
    MinPts then
23:             cell(p).isExclusive ← true; seeds.add(cell(p));
24:             C.IGs.add(innerCells(p));
25:         end if
26:     end for
27: end for
28: for each noise p ∈ Ns do
29:     get cell G ← p.coordinate;
30:     if |G^{Cs,Ns}| ≥ MinPts || Σ|innerCells(G)^{Cs,Ns}| ≥ MinPts || N_Eps(p) ≥ MinPts
    then
31:         G.isExclusive ← true; seeds.add(G);
32:         if G is an existing inclusive cell of Cs then
33:             getCluster(G).IGs.add(innerCells(p));
34:         else
35:             create new c-cluster C; C.IGs.add(innerCells(p));
36:         end if
37:     else if p falls in an existing inclusive cell on {Cs, Ns} then
38:         update the cell; delete p from Ns;
39:     end if
40: end for
41: //Second-round Merging
42: for each cell G ∈ seeds do
43:     Merge all c-clusters by Merging 1;
44:     Merge all c-clusters by Merging 2;
45: end for
```

6 Experimental Study

6.1 Experimental Setup and Methodologies

A comprehensive performance study has been conducted on 10 network-connected commodity computers deployed Hadoop platform to evaluate the effectiveness and efficiency of the proposed Cludoop algorithm. Each PC/node is equipped an Intel Core 2 Duo P8600 processor with 8GB memory, and runs a Ubuntu 11.4 operate system. One node is configured as both NameNode and JobTracker. Other nodes are configured as DataNode and TaskTracker.

Real Datasets. We use two real datasets in our experiments. The first real dataset is a trajectory database from $Geolife$ project [19] executed by Microsoft Research Asia. This dataset includes 178 users' 164,789 location points from May 7 to May 16, 2007. The second real dataset is $Taxi$ data, a real GPS trajectory data generated by 10,357 taxis in a period from Feb. 2 to Feb. 8, 2008 in Beijing [20]. The total number of points in this dataset is about 15 million, reaching at 1.46 GB size.

Synthetic Datasets. We also generate two synthetic datasets by our developed data generator using Matlab. The first synthetic dataset, denoted $Ssyn$, is generated to test the effectiveness of clustering, which includes multiple clusters of arbitrary shape comprised by 4413 points and 300 noises in 1000×1000 square.

The second large scale dataset $Lsyn$ is generated to evaluate performance of algorithm on processing big data, which contains about 2 billions data points in 2D space, reaching at 53.6GB size. For generality, we normalize all coordinates into range [0,1].

Alternative Algorithms. Our experiments focus on evaluating the effectiveness and efficiency of proposed Cludoop. For the effectiveness evaluation, we compare directly clustering results of our algorithm against the most wildly-used method DBSCAN. For the efficiency evaluation, we compare the CPU time utilized by our algorithm against the state-of-the-art method MR-DBSCAN as introduced in Sec. 2.

Metrics & Methodology. We measure the quality of clusters for effectiveness by Precision and Recall as follows: $Precision = \frac{R_c \cap D_c}{R_c}, Recall = \frac{R_c \cap D_c}{D_c}$, where D_c denotes the set of points labelled cluster ID or clusters obtained by DBSCAN, and R_c is the clustering result of our algorithm.

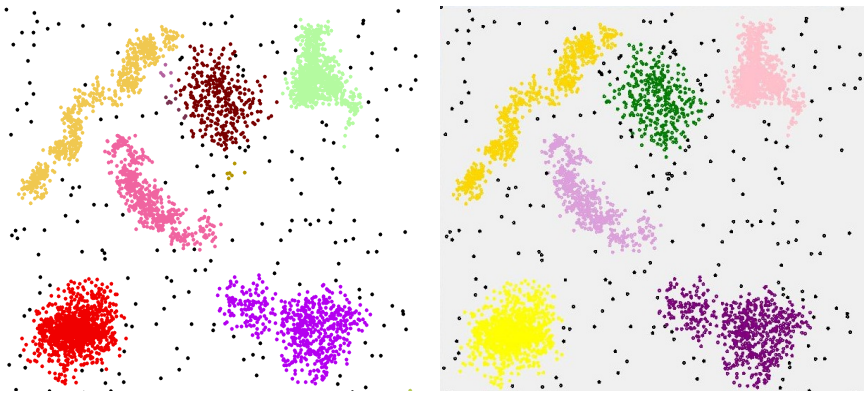
We also evaluate the performance of Cludoop method by varying the most important parameters. Specially, we measure the scalability of methods by varying the volume of dataset and the number of work node. Moreover, we also measure sensitivity of our Cludoop algorithm on efficiency by varying the input parameter Eps in a large range.

6.2 Effectiveness Evaluation

First, we evaluate the effectiveness of our proposed algorithm compared against DBSCAN using two evaluation metrics (visualization comparison, Precision/Recall) on Ssyn and GeoLife data.

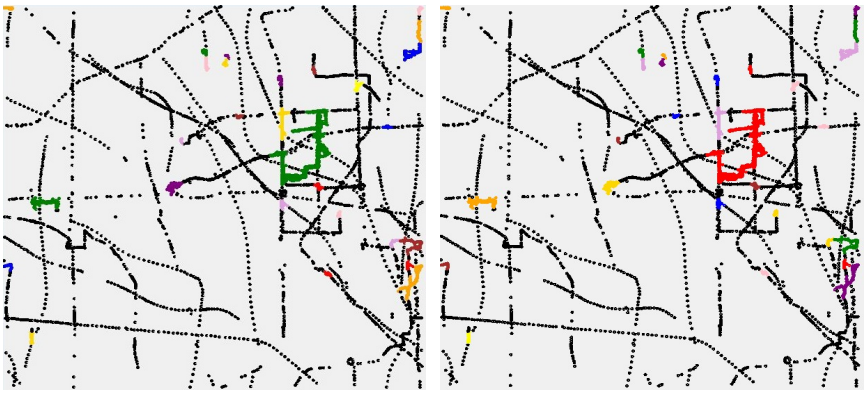
Fig. 4 shows the clustering result comparison of two algorithm on Ssyn data when Eps is fixed to 20 and $MinPts$ to 4. The black points are classified as noise. We can see that Cludoop can find almost same clusters as DBSCAN w.r.t. the same parameter setting. Only a very little of marginal border points are misclassified due to the proximate strategy employed for some border points in Reduce phase. Fig. 5 depicts the results of two algorithm on GeoLife data, setting Eps to 100 meters and $MinPts$ to 10. Again, our algorithm shows an excellent clustering ability, even better than that on Ssyn

data. This is because the locations of GeoLife is on or close to road networks, and only the location crowds at hot regions or crossings are classified as clusters, thus the ratio of border points not in inclusive cells is far below that of arbitrary distributed dataset.



(a) Cludoop (b) DBSCAN

Fig. 4 Visualization comparison on Ssyn data



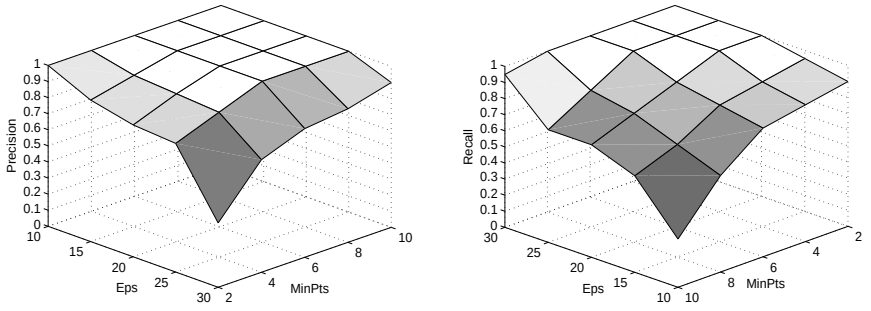
(a) Cludoop (b) DBSCAN

Fig. 5 Visualization comparison on GeoLife data

The Precision and Recall of Cludoop on Ssyn data are shown in Fig. 6. For Ssyn data, the Precision and Recall are computed based on labelled points. From Fig. 6(a), the Precision is nearly 100% once the the parameter *Eps* falls in a rather large range. Fig. 6(b) shows the Recall of our algorithm is also good in certain parameters space. For the real GeoLife data, we compute

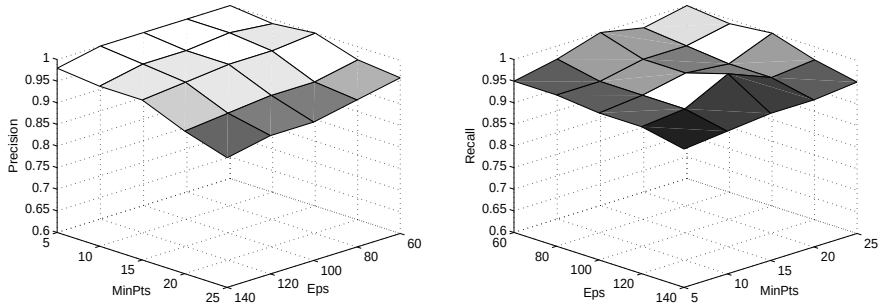
the Precision and Recall of Cludoop algorithm based on the clustering result of DBSCAN w.r.t. same parameters, excluding the noise. From Fig. 7, the Precision and Recall of clusters of Cludoop achieve on average 97% and 96% compared against DBSCAN in all test cases, respectively. In particular, Cludoop shows nearly 100% Precision when $Eps \leq 100$ and $MinPts$ varies form 5 to 20.

In summary the effectiveness study confirms that our distributed algorithm can obtain the correct clusters with noise under a loose parameter setting.



(a) Precision (b) Recall

Fig. 6 Effectiveness evaluation on Ssyn data



(a) Precision (b) Recall

Fig. 7 Effectiveness evaluation on GeoLife data

6.3 Efficiency Evaluation

Next we evaluate the efficiency of our algorithm compared against MR-DBSCAN algorithm using the Taxi and Lsyn data. We vary the most important parameters, to (1) assess the scalability of Cludoop versus MR-DBSCAN in term of efficiency, (2) evaluate sensitivity of parameter variation on our method.

6.3.1 Varying Volume of Dataset D

We first evaluate the scalability of our algorithm in term of the volume of dataset using Lsyn data. In this experiment we randomly extract four subsets from the Lsyn data from 20% to 80%. We fix the Eps to 0.001, $MinPts$ to 200. Fig. 8 shows the total running time of our algorithm and MR-DBSCAN on the five datasets. Cludoop algorithm exhibits much better scalability than MR-DBSCAN in term of CPU time. In particular, Cludoop saves on average 42% time compared to MR-DBSCAN on all tested cases. As the volume of dataset increases, both algorithms require more time to process this increased data points. However, our Cludoop algorithm saves more time with the rising of the number of data points. This is because more cells could be determined directly to be inclusive cells to reduce more distance calculations as the volume of D increases at fixed Eps and $MinPts$.

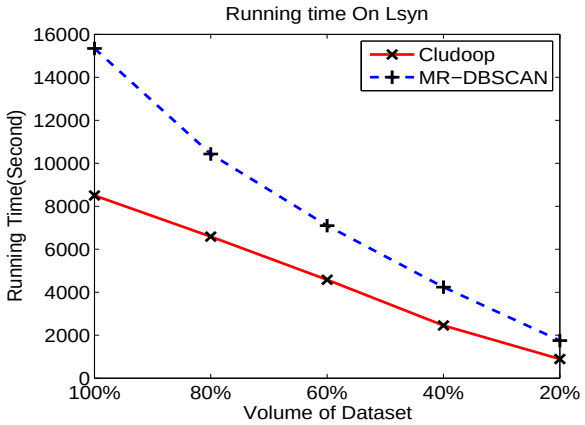


Fig. 8 Varying volume of dataset D on Lsyn data

6.3.2 Varying Number of Nodes nd

Next, we valuate the speedup of our algorithm as the number of nodes increases by varying work node from 2 to 9 on the Taxi data when Eps is fixed to 100 meters and $MinPts$ to 4. As shown in Fig. 9, Cludoop algorithm also clearly outperforms MR-DBSCAN in term of running time in all tested cases.

In particular, the speedup of our algorithm achieves at 6 times while that of MR-DBSCAN just reaches at 4 times when the number of nodes increases from 2 to 9. However, the speedup of our Cludoop also decreases as *nd* increases similar with MR-DBSCAN. This may be because that more border points or noise would be send to reducers in shuffle phase when launch more mappers, spending more network cost and merging time on reducers, although the increased parallel mappers reduce the pre-clustering time.

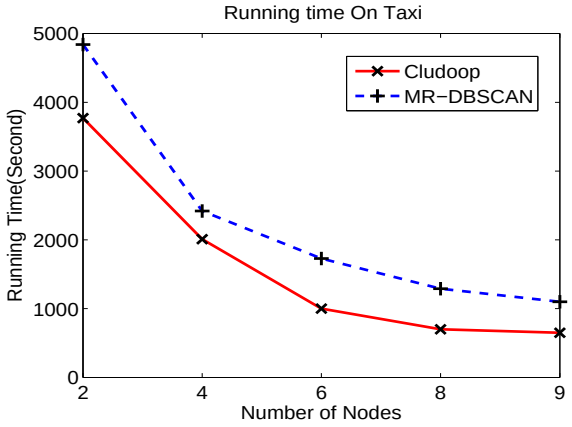


Fig. 9 Varying number of nodes *nd* on Taxi data

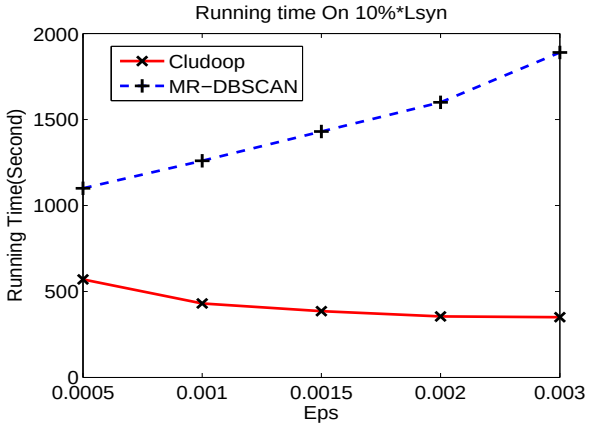


Fig. 10 Varying neighbor range *Eps* on 10%*Lsyn data

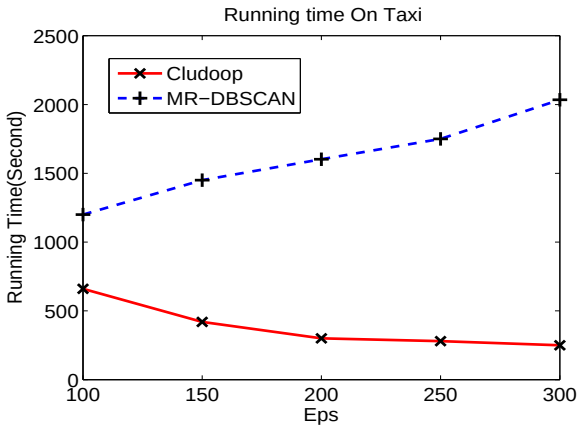


Fig. 11 Varying neighbor range *Eps* on Taxi data

6.3.3 Varying Neighbor Range Threshold Eps

Finally, we analyze the sensitivity of our algorithm w.r.t. the important parameter *Eps* on both 10%*Lsyn and Taxi datasets. We vary *Eps* from 0.0005 to 0.003 on the 10%*Lsyn data when $MinPts$ is fixed to 200, and vary *Eps* from 100 to 300 meters on Taxi data at fixed $MinPts = 4$. From Fig. 10 and Fig. 11, our proposed Cludoop shows outstanding superiority on time consumption compared to MR-DBSCAN w.r.t. *Eps*. This is because (1) our algorithm reduces the cost of maintaining cell index as *Eps* increases, and (2) more cells could be directly determined to be inclusive cells at a larger *Eps* value when $MinPts$ is fixed, saving much network cost in shuffle and merging time in reducer. Whereas larger *Eps* value leads to more duplicate points, costing more time in Shuffle and Reduce phase for MR-DBSCAN, and larger *Eps* is more likely to lead to imbalance workload in reducer due to its data partition is based on *Eps*. However, very large *Eps* value actually is meaningless for density-based clustering when $MinPts$ is fixed, since it can not return the meaningful clusters.

7 Conclusion

Density-based clustering for increasing big data applications is very important yet difficult task. This paper proposes an efficiency and load-balanced distributed density-based clustering for big data using existing data partition on Hadoop platform. Our algorithm incorporates a proposed serial clustering with c-cluster as plug-in on mapper, and a Merging-Refinement-Merging 3-steps framework to fast merge c-cluster on reducer. The empirical study on large-scale real and synthetic data shows that our Cludoop algorithm effectively finds the correct clusters and exhibits better scalability and efficiency than the state-of-the-art. An interesting direction for future work is to lever-

age optimized shuffle mechanism for further improving the workload balancing problem of clustering on Hadoop platform.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China (No.61403328, 61403329, 61302065, 61172049), the open project program of Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University (No.93K172014K13), the Shandong Provincial Natural Science Foundation (No. ZR2013FM011), and the project of Shandong Province higher educational science and technology program (No.J14LN24).

References

1. Martin Ester, Hans-Peter Kriegel, Jörg Sander et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining(SIGKDD)*. Portland, OR, 1996.
2. M. Ankerst , M. Breunig, H.-P. Kriegel ,and J. Sander. OPTICS: Ordering Points to identify the clustering structure, *Proc. Of ACM SIGMOD International Conference on Management of Data(SIGMOD)*, Philadelphia, PA, 1999
3. Alexander Hinneburg, Daniel A. Keim. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. *Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining(SIGKDD)*, New York, pp.58-65, 1998.
4. Kyuseok Shim. MapReduce Algorithm for Big Data Analysis. *Proc. of the International Conference on Very Large Data Bases (VLDB)*, Istanbul, pp.2016-2017, 2012.
5. Jörg Sander, Martin Ester, Hans-Peter Kriegel, Xiaowei Xu. 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
6. Xiaowei Xu, Jochen Jäger, Hans-Peter Kriegel. A Fast Parallel Clustering Algorithm for Large Spatial Databases. *Data Mining and Knowledge Discovery*, vol.3, no.3, pp. 263-290, 1999.
7. Eshref Januzaj, Hans-Peter Kriegel, Martin Pfeifle. DBDC: Density Based Distributed Clustering. *Proc. of International Conference on Extending Database Technology(EDBT)*, Heraklion, pp:88-105, 2004
8. Eshref Januzaj, Hans-Peter Kriegel, Martin Pfeifle. Scalable Density-Based Distributed Clustering. *Proc. of 8th European Conference on Principles and Practice of Knowledge Discovery in Databases(PKDD)*, Heraklion, pp:88-105, 2004
9. Manoranjan Dash, Simona Petrutiu, Peter Scheuermann. Ecient Parallel Hierarchical Clustering. *Euro-Par 2004 Parallel Processing, Lecture Notes in Computer Science*, Volume 3149, pp. 363-371, 2004
10. Stefan Brecheisen, Hans-Peter Kriegel, and Martin Pfeifle. Parallel Density-Based Clustering of Complex Objects. *Proc. of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Berlin Herdelberg, pp:179-188, 2006
11. Christian Böhm, Robert Noll, Claudia Plant, Biauca Wackersreuther, Andrew Zherdin. Data Mining Using Graphics Processing Units. *Transactions on Large-Scale Data- and Knowledge-Centered Systems I, Lecture Notes in Computer Science*, Volume 5740, pp:63-90, 2009
12. Christian Böhm, Robert Noll, Claudia Plant, Biauca Wackersreuther. Density-Based Clustering using Graphics Processors. *Proc. of ACM International Conference on Information and Knowledge Management (CIKM)*, Hong Kong, pp:661-670, 2009

13. Guilherme Andrade, Gabriel Ramos, Daniel Madeira, Rafael Sachetto, Renato Ferreira, Leonardo Roch. G-DBSCAN: A GPU Accelerated Algorithm for Density-based Clustering. *Proc. of International Conference on Computational Science (ICCS) 2013*, Barcelona, Spain, pp:369-378, 2013
14. Alina Ene, Sungjin Im, Benjamin Moseley. Fast Clustering using MapReduce. *Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining(SIGKDD)*, San Diego, pp:681-689, 2011
15. Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao et al. MR-DBSCAN: An Efficient Parallel Density-based Clustering Algorithm using MapReduce. *Proc. of IEEE 17th International Conference on Parallel and Distributed Systems(ICPADS)*, Tainan, pp:473-480, 2011
16. Bi-Ru Dai, I-Chang Lin. Efficient Map/Reduce-based DBSCAN Algorithm with Optimized Data Partition. *Proc. of 5th International Conference on Cloud Computing*, Honolulu, pp:59-66, 2012
17. Yaobin He, Haoyu Tan, Wuman Luo, Shengzhong Feng, Jianping Fan. MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for Heavily skewed data. *Frontiers of Computer Science*, Vol. 8, No. 1, pp:83-99, 2014
18. Robson L. F. Cordeiro, Caetano Traina Jr., Agma J. M. Traina et al. Clustering Very Large Multi-dimensional Datasets with MapReduce. *Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining(SIGKDD)*, San Diego, pp:690-698, 2011
19. Yu Zheng, Xing Xie, and W.-Y. Ma. Geolife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Engineering Bulletin*, vol.33, no.2, pp:32-40, 2010
20. Jing Yuan, Yu Zheng, Xing Xie, and G. Sun. T-Drive: Enhancing Driving Directions with Taxi Drivers' Intelligence. *IEEE Transactions on Knowledge and Data Engineering*, vol.25, no.1, pp:220-232, 2013