Journal of Circuits, Systems, and Computers
Vol. 28, No. 4 (2019) 1950065 (26 pages)

DOI: 10.1142/S0218126619500658

DHC: A Distributed Hierarchical Clustering Algorithm for Large Datasets*

Wei Zhang[†,‡], Gongxuan Zhang[†,§], Xiaohui Chen[†], Yueqi Liu[†], Xiumin Zhou[†] and Junlong Zhou[†]

[†]*Computer Science and Engineering, Nanjing University of Science and Technology, NO.200 Xiaolingwei Road, Nanjing 210094, P. R. China*

[‡]*Computer Science and Technology, Huaiyin Normal University, NO.111 Changjiangxi Road, Huai'an 223300, P. R. China*

[§]*gongxuan@njust.edu.cn*

Received 30 January 2018
Accepted 31 May 2018
Published 4 July 2018

Hierarchical clustering is a classical method to provide a hierarchical representation for the purpose of data analysis. However, in practical applications, it is difficult to deal with massive datasets due to their high computation complexity. To overcome this challenge, this paper presents a novel distributed storage and computation hierarchical clustering algorithm, which has a lower time complexity than the standard hierarchical clustering algorithms. Our proposed approach is suitable for hierarchical clustering on massive datasets, which has the following advantages. First, the algorithm is able to store massive dataset exceeding the main memory space by using distributed storage nodes. Second, the algorithm is able to efficiently process nearest neighbor searching along parallel lines by using distributed computation at each node. Extensive experiments are carried out to validate the effectiveness of the DHC algorithm. Experimental results demonstrate that the algorithm is 10 times faster than the standard hierarchical clustering algorithm, which is an effective and flexible distributed algorithm of hierarchical clustering for massive datasets.

Keywords: Hierarchical clustering; nearest neighbor search; distributed storage and computation; data mining.

1. Introduction

Hierarchical clustering has been used in many fields, especially in machine learning and data mining. The basic approaches to hierarchical clustering, agglomerative and

*This paper was recommended by Regional Editor Tongquan Wei.
§Corresponding author.

divisive, are generating hierarchical clusters of datasets. However, hierarchical clustering has a major difficulty of searching *nearest neighbor* (NN). A standard agglomerative hierarchical clustering (AHC)[1] algorithm begins with n original clusters of an Euclidean space, and has the time complexity of $O(n^3)$ and the space complexity of $O(n^2)$.

Numerous hierarchical clustering algorithms have been proposed to reduce the time and space complexity of hierarchical clustering, such as SLINK[2] and CLINK.[3] Nearest-neighbor chains (NNC)[4] algorithm reduces time complexity of the distance calculations to $O(n^2)$ by using a stack that maintains the set of active clusters. Multi-level hierarchical clustering (MLHC)[5] algorithm provides an order-N speedup over the standard AHC with the time overhead of $O(n^2 \log n)$. However, these algorithms are not suitable for handling large amounts of data.

Considerable investigations have been devoted to the NN searching algorithm. They mainly focus on improving the efficiency of the NN searching, reducing the neighbor search time, and dealing with massive datasets. K-nr[6] is a novel framework for approximate proximity search algorithms which introduce a space-efficient representation data structures called *indexes* for searching very large databases by using the main memory. An LC indexing algorithm[7] is proposed to KNN and range queries, accelerated on the Inter Xeon Phi coprocessor. In our previous work,[8] we proposed an efficient NN search method called *nearest-neighbor boundary* (NNB) that reduces time complexity of the NN search by using quad-tree[9–11] or kd-tree[12,13] data structures for data partition. However, these methods are difficult to be extended when dealing with data exceeding the main memory space.

In practice, some hierarchical clustering algorithms are proposed to accelerate the computation speed. A sensitivity analysis of algorithms[14] to segment and classify high-resolution images, which attain a parallel efficiency by using the CPUs and the Phi available on 256 nodes. Based on hierarchical inter-class structures, a novel image classification method[15] for multi-class classification is accelerated by using a fast algorithm to compute the similarity metric between categories on GPUs. A hierarchical cluster guided labeling[16] is designed to efficiently collect a diverse set of labeled training data onto a single human annotator. Genie[17] is a new hierarchical clustering linkage criterion and is easily parallelizable to speed up its execution. These algorithms try to deal with different applications for large datasets.

In this paper, to flexibly deal with massive data onto the main memory space, we implement a distributed storage and computation hierarchical clustering algorithm based on NNB called *distributed hierarchical clustering* (DHC). As compared to the above introduced techniques,[5,7,14–16,18] DHC has the following two advantages. The first advantage of DHC is that it is able to store massive datasets by using distributed storage nodes. The second advantage of DHC is that it is able to efficiently process the NN searching by distributed computation at each storage node. Therefore, DHC is a flexible distributed algorithm of hierarchical clustering for massive datasets.

1.1. *Motivation*

Hierarchical clustering (also called Hierarchical Cluster Analysis or HCA) is a cluster analysis method which builds a hierarchy of clusters to find relationships by clustering items as an integral component. Clustering method groups data onto several parts using some measurements such that the items in the same cluster are rather similar than the items in different clusters. For example, if given a set position of friends, HCA divides them into some groups where every member of the group is close to each other, as described in Example 1.

Example 1. Agglomerative is a "bottom up" approach: each cluster starts in its own cluster, and pairs of clusters are merged as one new cluster. The record of this newly merged cluster represents a hierarchy of clusters. Its process includes the following steps and gets a dendrogram of HCA, as shown in Fig. 1.

(1) There are 9 persons in Fig. 1(a). Hierarchical clustering assigns each person as the original point of its own cluster.
(2) It calculates the similarity between all cluster pairs and finds the most close cluster pair that is closest to each other such as A and B.
(3) The algorithm merges the most close pair A and B into a new cluster J, as shown in Fig. 1(b).
(4) It repeats the process until only one cluster Q is left, as shown in Fig. 1(c).

At each repetition of clustering, it records each cluster pair that has been merged. This merging record represents a dendrogram that merges the original points of upper-level clusters and finally reaches to the root, as shown in Fig. 1(d). From the dendrogram, the relative of persons is expressed clearly and the persons are divided into different clusters naturally. There are many clustering algorithms[1] such as BIRCH,[19] CMUNE,[20] CURE[21] and Chameleon[22]; each has different cost and accuracy. Hierarchical clustering is one of the most basic and most accurate clustering algorithms. However, the complexity of searching the NN of each one is $O(n^2)$ at each repetition and the complexity of hierarchical clustering is $O(n^3)$.

Because the complexity of AHC algorithm is highly dependent on the complexity of searching the NN of each one, so search for the NN is a key technique of clustering. If we reduce the time complexity of the NN searching, we can reduce the time complexity of hierarchical clustering. Partitioning is a key technology of clustering large datasets to reduce the complexity of searching. After partition, the times of finding the closest cluster pairs decrease obviously, as shown in Example 2.

Example 2. Partition reduces the time complexity of searching the NN and time complexity of agglomerative clustering, as shown in Fig. 2.

(1) In the original AHC, 9 persons are in Fig. 2(a). For person I, 8 times are needed to calculate the distance between others, then total $8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 36$ times of distance calculation of all pairs are needed to get the most close pair.

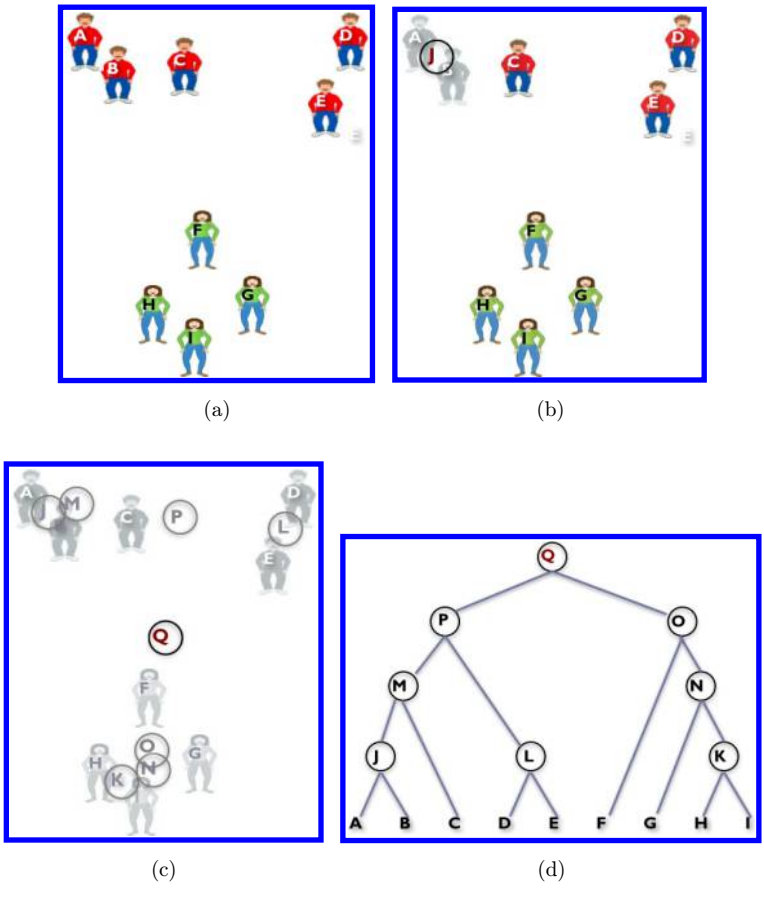


Fig. 1. An illustrative example of clustering process: (a) the original point A to I; (b) the closet pair A and B is merged to a new cluster J; (c) only one cluster Q is left after repetition; (d) the dendrogram of Hierarchical Cluster Analysis represents the result of the process.

(2) After partitioned into three regions, the persons in one region are closer to each other than in the other regions, as shown in Fig. 2(b). In the three regions, there are 3 persons, 2 persons and 4 persons in different regions. So, $3+2+1=6$, $2+1=3$ and $4+3+2+1=10$ times of distance calculation are needed to get the most close pairs in the three regions. The total distance calculation times is $6+3+10=19$.

As described above, we can get that partitioning is the basic technology of clustering large datasets, and distributed storage and computation are the essential techniques of clustering large datasets. In this paper, we first introduce the definition of the NNB and then use it to partition a large dataset. The utilization of NNB can construct the distributed storage and computation framework of hierarchical clustering.

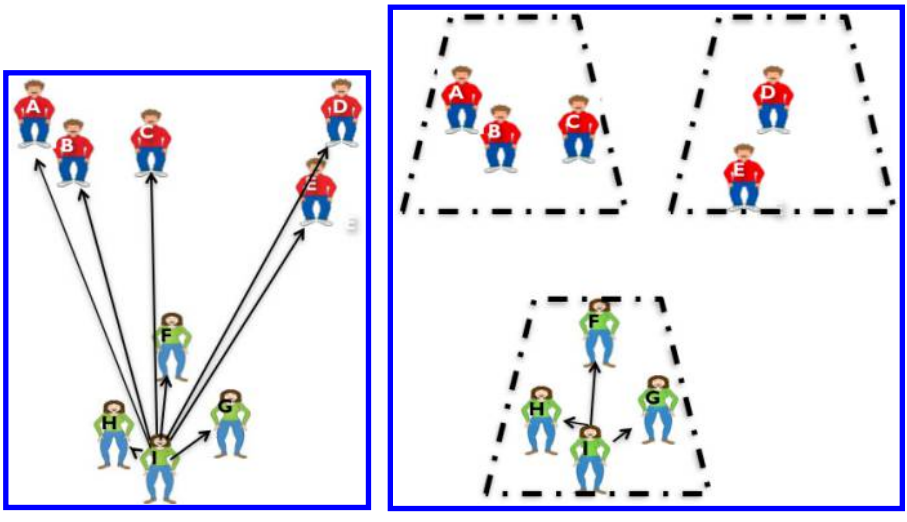


Fig. 2. For finding the NN: (a) distance calculation times of person I to others are 8, (b) after the partition, person I only needs to be calculated the distance from others in the region, the number of calculations is 3.

1.2. *Contributions*

In our previous work,[8] we proposed to use NNB[8] to accelerated hierarchical clustering in the serial algorithm. The serial algorithm is limited by storage capacity and computation power of single computer. In this paper, we propose a novel distributed storage and computation hierarchical clustering algorithm DHC to overcome the shortage of our previous serial algorithm. DHC is based on NNB for distributed storage and parallel computing of large datasets. Based on NNB, DHC first effectively divides large datasets into small subsets, then searches the NN with small subsets and gets the global mutual nearest-neighbor (MN) at last, as shown in Fig. 3.

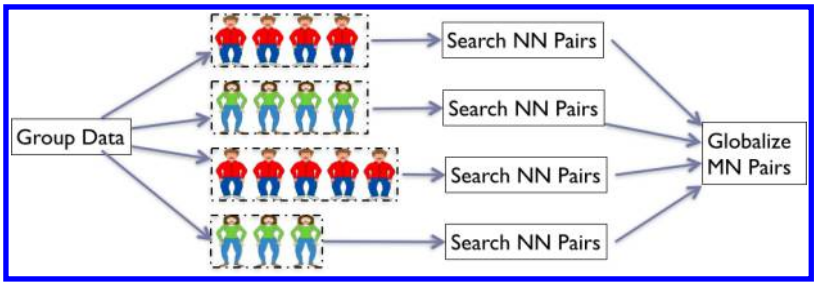


Fig. 3. DHC based on NNB.

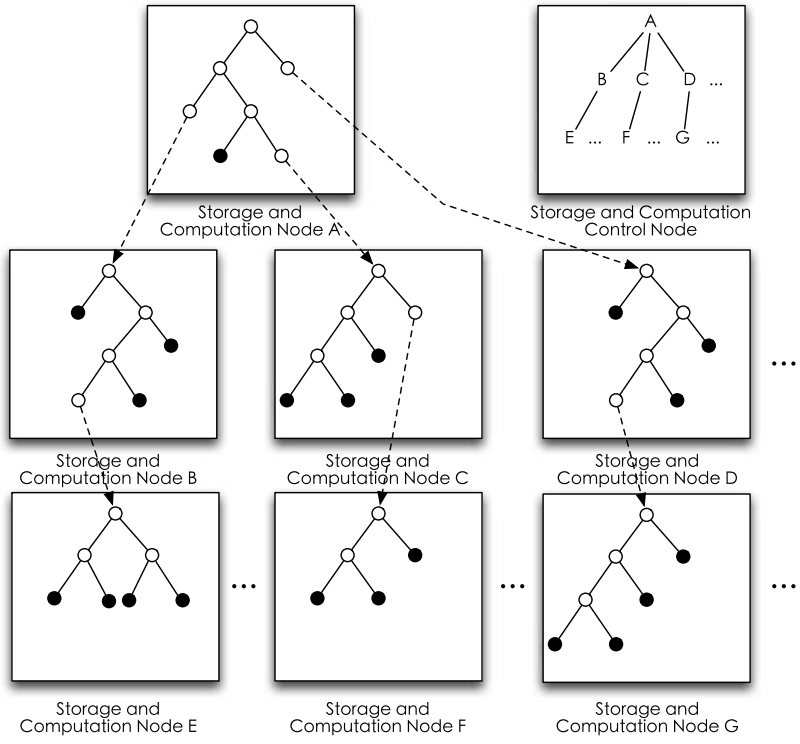


Fig. 4. Distributed storage and computation framework for hierarchical clustering.

In distributed computation, file operations of distributed file system and constant IO operations incur high IO and distributed node communication overhead. In our work, distributed memory storage and a control node is employed to reduce the communication between the computation nodes, as shown in Fig. 4.

Using new distributed storage and computation concept, we get a flexible hierarchical clustering algorithm for large datasets. In addition, the new algorithm is easy to be deployed on computer clusters. The major contributions to this paper are summarized as follows:

(1) We propose distributed quad-tree[9–11] or kd-trees[12,13] to store high-dimensional datasets exceeding the main memory space. Quad-tree and Kd-tree are important data structures and space-partitioning data structures for organizing points in a two-dimension or k-dimension space efficiently. We construct quad-tree or kd-trees on distributed node to store massive datasets.
(2) We present a fast clustering distributed storage and computation algorithm DHC based on NNB. NNB is a region in which a point NN must in the scope of the region. We use the NNB to group the points of calculating the NNs on each storage node along parallel lines.

(3) We evaluate the performance of DHC under different parameters, such as size of quad-tree of kd-tree leaf node and number of storage nodes etc. We present quite an amount of experimental studies to prove the effectiveness of the DHC algorithm.

The rest of the paper is structured as follows: Section 2 gives the related works on DHC. Section 3 presents the theoretical proof of data partition by NNB. Section 4 describes the concept of DHC algorithm. In Sec. 5, we describe the evaluation results of our proposals. Finally, Sec. 6 concludes the paper.

2. Related Work

Many efforts of research for improving hierarchical clustering algorithm performance have been conducted from different aspects. DHC algorithm focuses on improving the efficiency of the NN searching. The key idea of the DHC algorithm for greatly reducing the number of distance computations required for clustering are to use the quad-tree[9–11] or kd-tree[12,13] for calculating NNB of each region.

2.1. *Clustering for large datasets*

In traditional studies of hierarchical clustering algorithms, the focus on this research is on calculating the similarity between clusters. However, when the dataset becomes very large, the computation of these clustering algorithms becomes time-consuming.

Some methods, such as CMune,[20] Canopies[23] and CURE,[21] divide the dataset into small subsets for improving the performance. But the results of those clustering methods are greatly dependent on the selection of parameters. CMune[20] is a variation on the Shared Nearest Neighbor algorithm, which selects the best parameter number of the neighborhoods of a point for good performance. Canopies[23] efficiently divides data into some overlapping subsets by using a cheap approximate distance measure. CURE[21] represents clusters by using a certain number of scattered points of the clusters and shrinks those points toward the center of the clusters by using a specified fraction. All those algorithms use different partition methods to accelerate computation speed while those partition methods cannot ensure finding the NN of each point and different parameters' selections affect the clustering result. Motivated by NNB, the partition method of our DHC algorithm can find the NN of each point and get the same clustering result as traditional studies of hierarchical clustering algorithms AHC,[1] while it just requires split threshold for performance adjustment, as show in Table 1.

Recently, some parallel and disturbed computing frameworks were used to speed up the performance of clustering. IncDiSC[24] formulates the single-linkage hierarchical clustering problem as a Minimum Spanning Tree (MST) construction problem on a complete graph and implement the unified algorithm by employing MapReduce framework. Shortest Path Betweenness MapReduce Algorithm (SPB-MRA)[25] is a

Table 1. The influence of algorithms parameter over performance and accuracy.

Algorithm	Parameter	Performance	Accuracy
CMune[20]	Number of the neighborhoods	Yes	Yes
Canopies[23]	Approximate distance measure	Yes	Yes
CURE[21]	Number of scattered points	Yes	Yes
DHC	Split threshold	Yes	No

parallel version of a divisive hierarchical clustering algorithm for community detection based on the MapReduce model. Sun *et al.*,[26] presents an efficient hierarchical clustering method of mining large datasets with MapReduce which includes two optimization techniques: Batch updating to reduce the computational time and communication cost among cluster nodes, and co-occurrence-based feature selection to decrease the dimension of feature vectors and eliminate noise features. But in MapReduce, the file operations of distributed file system and constant disk IO operations incur high disk IO and distributed node communication overhead. In our work, DHC employs a distributed memory storage and a Control Node to reduce the IO operations on computation nodes.

2.2. *Distance measure*

Measuring the distance between objects is the foundation of clustering algorithms since all of these algorithms are built on the similarity between objects. The common distance measure used here is Euclidean distance. The similarity-based linkage function is an important measurement of similarity comparison between two clusters of hierarchical clustering. The common linkage functions are shown in Fig. 5.

Single link or MIN, Complete link or MAX, Croup average, Centroid method and Ward's method are general linkage functions.[27,33–35] Linkage function of Single link is the minimum distance of a pair of objects between cluster A and cluster B. The function of the MIN linkage is expressed as

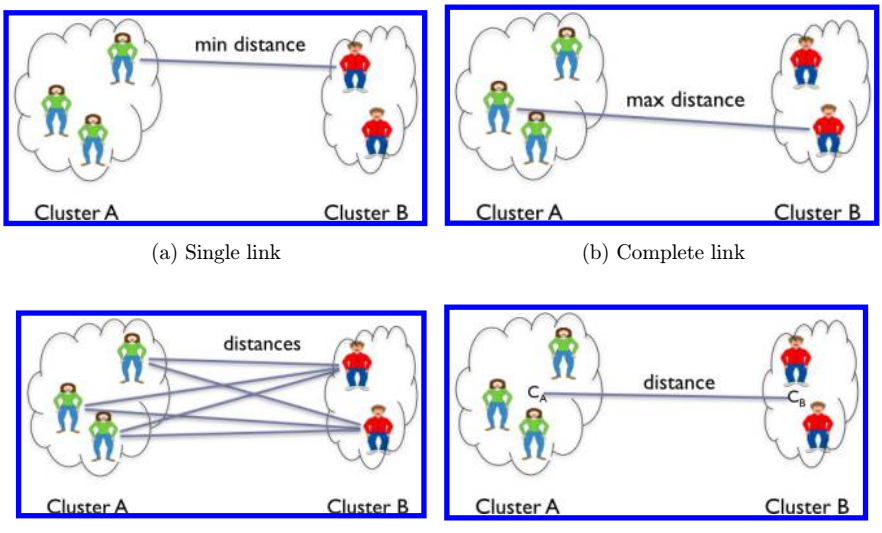
$$\text{Link}_{\min}(A, B) = \min_{x \in A, y \in B} \text{distance}(x, y)\,, \tag{1}$$

where distance(x, y) is Euclidean distance. Linkage function of Complete link is the maximum distance of a pair of objects between cluster A and cluster B. The function of the MAX linkage is formulated as

$$\text{Link}_{\max}(A, B) = \max_{x \in A, y \in B} \text{distance}(x, y)\,, \tag{2}$$

where distance(x, y) is the Euclidean distance. Linkage function of Group average is the average distance of each pair of objects between cluster A and cluster B. The function of the Croup average linkage is written as

$$\text{Link}_{\text{avg}}(A, B) = \sum_{x \in A, y \in B} \frac{\text{distance}(x, y)}{n_A * n_B}\,, \tag{3}$$



(a) Single link (b) Complete link

(c) Group average (d) Centroid method

Fig. 5. Common linkage functions.

where n_A and n_B denote elements number of clusters A and B. Linkage function of Centroid method is the average distance of each pair of objects between cluster A and cluster B. The function of the Centroid method linkage is described as

$$\text{Link}_{\text{centroid}}(A, B) = \text{distance}(c_A, c_B)\,, \tag{4}$$

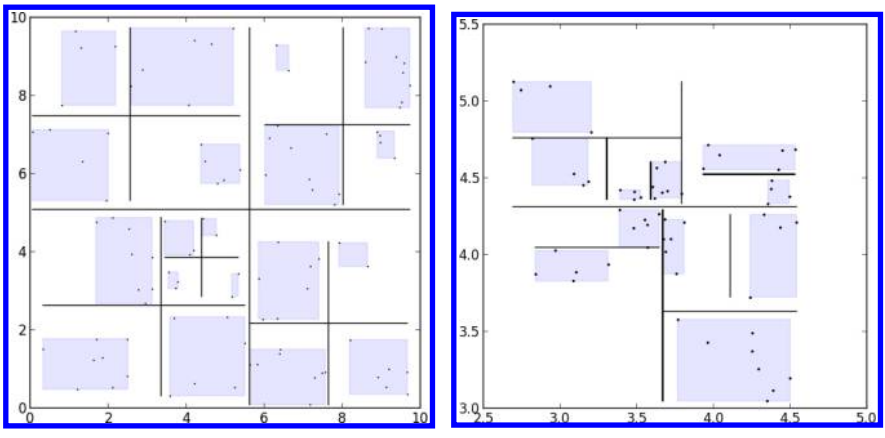
where c_A and c_B denote the centroids of clusters A and B. Ward's method is another common linkage function to measure the distance between each pair of objects of cluster A and cluster B. The function of the Ward's method linkage is

$$\text{Link}_{\text{ward}}(A, B) = \frac{\text{distance}(c_A, c_B)}{\frac{1}{n_A} + \frac{1}{n_B}}\,, \tag{5}$$

where distance(x, y) is the Euclidean distance, n_A and n_B denote elements number of clusters A and B, c_A and c_B denote the centeroids of clusters A and B.

2.3. *Quad-tree and Kd-tree*

Quad-tree[9–11] and kd-tree[12,13] are important data structures in computer science. Those space-partitioning data structures organize points by a two-dimensional or k-dimension space. Quad-trees are tree data structures that consist of internal node which has exactly four children in two-dimensions, as shown in Fig. 6(a).[8] Quad-trees are usually used to partition a two-dimensional space in which each internal node is recursively subdivided it into four quadrants or regions. Kd-trees are binary trees which have an internal node as a k-dimensional point, as shown in Fig. 6(b).[8]



(a) A quad-tree example (b) A kd-tree example

Fig. 6. Examples of quad-tree and kd-tree.

Every nonleaf node in kd-trees are divided into two parts by using a splitting hyperplane.

The forms of quad-trees and kd-trees have some common features:

(1) They divide space by adaptable cells or splitting planes.
(2) Each cell is limited by a maximum capacity. When a cell is beyond maximum capacity, the cell splits and forms splitting planes.
(3) The tree is decomposed as the spatial directory.

According to the type of data they represent, including areas, points, lines and curves, quad-trees can be classified into different types. Quad-trees can also be classified by whether the shape of the tree is independent of the order data is processed. Quad-trees is very efficient in comparing two-dimensional ordered data points in $O(\log n)$ time. Kd-trees data structure and search algorithms are the generalization of classical binary search trees to higher dimensional spaces. Based on kd-trees, the time complexity is $O(\log n)$ for locating the NN of a point.

Clustering high-dimensional data is the cluster analysis of data with anywhere from a few dozen to many thousands of dimensions. Clustering in high-dimensional data has a number of unintuitive properties that are referred to as the *curse of dimensionality*.[36] Because the distance between any two points in a given dataset converges, the concept of distance becomes less precise as the number of dimensions grows. Thus, the discrimination of the nearest and farthest point in particular becomes meaningless.

In our algorithm, kd-trees[12,13] are employed to partition data structures for organizing points in k-dimensional spaces. We extend kd-trees for clustering in high-dimensional spaces.

3. Data Partition and Globalization

As described above, data partition is the basic technology of clustering large datasets. Distributed storage and computation are the essential techniques of clustering large datasets. We first introduce the NNB, then discuss the use of NNB for finding the NNs in partition dataset.

3.1. *Nearest neighbor boundary*

NNB is a region in which a point's NN must be in the scope of the region. Given points $A(x_a, y_a), B(x_b, y_b)$ and $C(x_c, y_c)$, define

$$\text{Scope}(A) = \{C(x, y) | \text{distance}(A, C) \leq \text{distance}(A, B)\}, \tag{6}$$

where distance(A, B) denotes the Euclidean distance between A and B, distance (A, C) denotes the Euclidean distance between A and C. The two Euclidean distances are given by

$$\text{distance}(A, B) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}, \tag{7}$$

$$\text{distance}(A, C) = \sqrt{(x_a - x_c)^2 + (y_a - y_c)^2}. \tag{8}$$

The Scope(A) is a neighbor boundary of point A. Based on the NN concept, we have the following properties.

Property 1. Given two points $A, B \in \text{Scope}(A)$, if $\exists C$ is the NN of A, then distance$(A, B) \geqslant$ distance(A, C).

From Property 1, if distance$(A, B) <$ distance(A, C), then C is not A's NN. It means that point A has a neighbor B and the NN of A must be in Scope(A). Based on Property 1, the definition of NNB is given as follows:

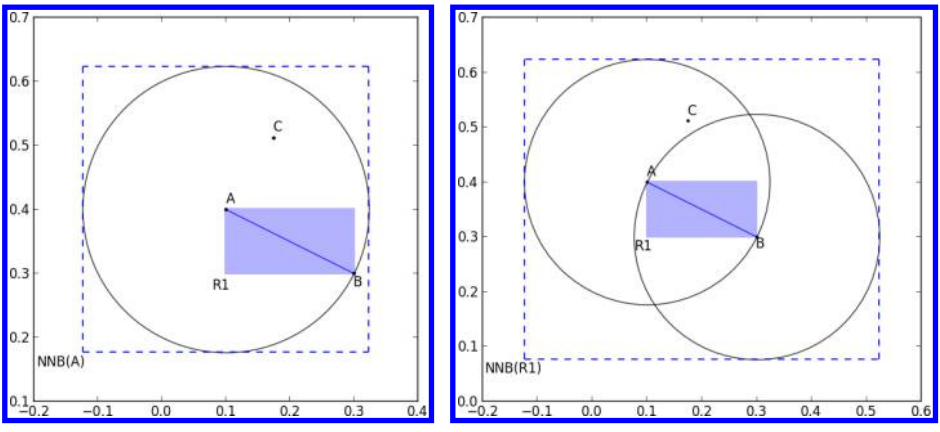
Definition 1. Given two points A, B, NNB of point A is a rectangle region defined as follows: NNB$(A) = \{(x, y) | x_a - \text{distance}(A, B) \leq x \leq x_a + \text{distance}(A, B),\ y_a - \text{distance}(A, B) \leq y \leq y_a + \text{distance}(A, B)\}$.

From Definition 1, if the NN of A exists in the regions, the scope of A's NN is a circle whose radius is distance(A, B), as shown in Fig. 7(a).[8] Based on Definition 1, the property of NNB is given as follows:

Property 2. NNB(A) contains the A's NN scope Scope(A).

From Property 2, if A's NN exists, it should be in NNB(A). Because NNB(A) includes Scope(A). Based on the NNB definition of point A, the NNB definition of region R_1 is given as follows.

Definition 2. Give the region R_1, the NNB of region R_1 is a rectangle region defined as follows: NNB$(R_1) = \{(x, y) | x_a - \text{distance}(A, B) \leq x \leq x_b + \text{distance}(A, B),\ y_a - \text{distance}(A, B) \leq y \leq y_b + \text{distance}(A, B)\}$.



(a) (b)

Fig. 7. (a) The point A's NN scope and NNB. (b) The region R_1's NN scope and NNB.

From Definition 2, given two points $A, B \in$ region R_1, it is known that NNB(R_1) includes Scope(A) and Scope(B). Based on Definition 2, the property between NNB of region and NNB of points is given as follows:

Property 3. Given a region R_1 and a point $A \in R_1$, NNB(R_1) must contain NNB(A).

Property 3 indicates that all NN scopes of region R_1 points are included in NNB (R_1) as shown in Fig. 7(b).[8]

3.2. *MN globalization*

In our algorithm, MN[5] is an important concept to ensure the correctness of the distributed storage and computation algorithm. The concept of MN means that if A's NN is B and A's NN is B too, then A and B are a pair of MN. All pairs of MNs can be found in an NNB of a partition dataset called local mutual nearest-neighbors (LMN), but the LMNs in an NNB may not be a global mutual nearest-neighbor (GMN). The GMN means that two points are the NN of each other in the whole dataset. To deal with the problem of finding global nearest neighbors, (GNN), pairs of MN will be processed by different methods. An example is described as follows:

Example 3. As an example, as shown in Fig. 8,[8] to obtain GMN pairs is described as follows:

Point A and F are both in the region R_1, point A is the NN of point F and point F is the NN of point A, so point A and F are the LMN pair and also the GMN pair.

Point B and D are not in the same region. Point B that is in the region R_2 and in NNB scope of region R_1 is the NN point of point D in the region R_1. Point B and D are the LMN pair in NNB scope of the region R_1. Point D that is in the region

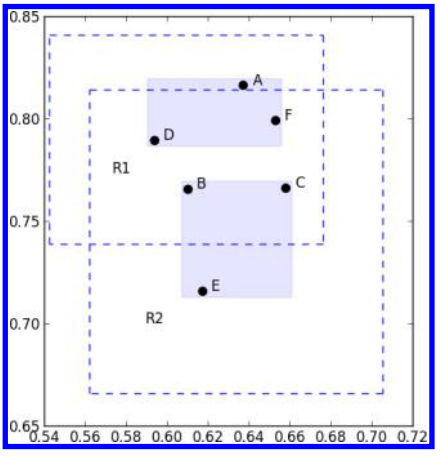


Fig. 8. An example of MNs search. Points B and D are the MN pair and Points C and F are not the MN pair.

R_1 and in NNB scope of the region R_2 is the NN point of point B in the region R_2. Points D and B are a LMN pair in NNB scope of the region R_2. So Point B and D are a GMN pair.

Point F and C are not in the same region. Point F that is in the region R_1 and in NNB scope of the region R_2 is a NN point of point C in the region R_2. Point F and C are the LMN pair in NNB scope of region R_2. Point C that is in the region R_2 and in NNB scope of the region R_1 is not the NN point of point F in the region R_1. Point C and F are not a LMN pair in the NNB scope of region R_1 So, Points C and F are not the GMN pair.

This example shows a process of finding the GMN pairs from the dataset. Based on example 3, some observations are described as follows:

Observation 1. If two points are both inner points of a region and the two points are a pair of MN, we get that they are a pair of GMN, where the inner points of a region R means the points are in the region.

Observation 2. If one point is an inner point and the other is an outer point of a region, and there are a pair of MN in this region, we get that the outer point is the GNN of the inner point. If the inner point is the GNN of the outer point too, we get that they are a pair of GMN, where the outer point means the point is outside of the region but in the NNB scope of the region.

Based on the following above observations, the MN globalization process has followed steps: At the first step, we search LMN pairs in each NNB of a partition dataset; at the second step, we get GMN pairs and GNN pairs from the LMN pairs which are the results of the first step; at the last step, we get GMN pairs for the GNN pairs from pairs which are produced in the second step.

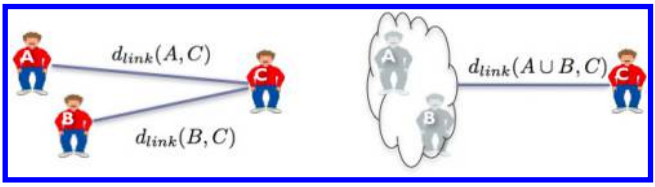


Fig. 9. Distances in cluster aggregate inequality.

3.3. *Algorithm correctness*

In our algorithm, *Cluster Aggregate Inequality* and MN are important concepts to ensure the correctness of the algorithm. If a linkage function Link() is reducible, it satisfies *Cluster Aggregate Inequality*, Eq. (9), as shown in Fig. 9,

$$\text{Link}(A \cup B, C) \geqslant \min(\text{Link}(A, C),\ \text{Link}(B, C))\,. \tag{9}$$

Single link, complete link, group average and Ward's method satisfy *Cluster Aggregate Inequality*.[33,5]

MN indicates that if A's NN is B and B's NN is A, then A and B are a pair of MN. The linkage function must satisfy the *Cluster Aggregate Inequality*,[33,5] after merging MN-pairs in parallel, we can get the same clustering results as the results that are produced by the standard AHC,[4,5] Ward's method[34] is employed in our algorithm as the linkage function and Ward's method satisfies the *Cluster Aggregate Inequality*.[33,5] After the DHC algorithm finding all MNs and merging them in each iteration, the DHC algorithm possesses the same clustering results as the results produced by the AHC algorithm.

4. The DHC Algorithm

In this section, we delineate the algorithm procedure of DHC. The important parts of DHC algorithm are NNB calculation and distributed storage and computation.

4.1. *The process of algorithm*

Based on kd-tree and MN technology, the DHC algorithm consists of five steps in each iteration. First, we partition the input dataset by using kd-trees. Then, we group the data by calculating NNB of each region in kd-tree. We search MN pairs and NN pairs in all data groups. We find all GMN pairs from the NN pairs. Last, we merge those GMN pairs from the results of the fourth step to new clusters. The proposed DHC clustering algorithm process is shown as Fig. 10.

4.2. *NNB calculation*

NNB calculation is an important step in the DHC algorithm. Based on the definition of NNB, all NNBs of points in a region must be calculated firstly and then the NNB

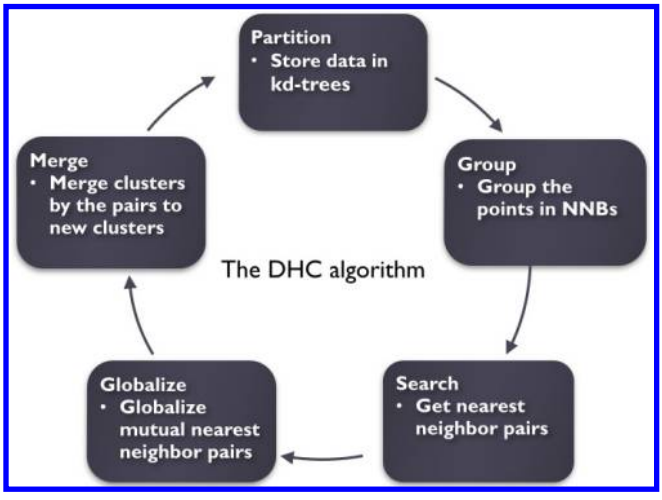
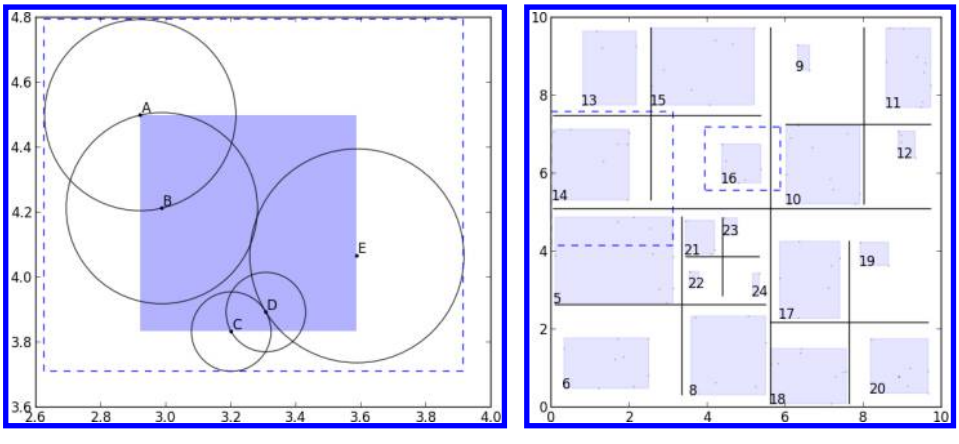


Fig. 10. The process of DHC algorithm.

of the region will be computed from all NNBs of points in the region. There is an example that region 16 has 5 points (A to E). The NNBs of the 5 points are 5 circles and the NNB of the region is a rectangle which just includes the 5 circles, as shown in Fig. 11(a).[8]

From Fig. 11(b), we observe that the NNB of region 16 is a necessary boundary for searching MN of each point and the NNB reduces the NN searching scope in the region NNB, as shown in Fig. 11(b). The pseudo-code of region NNB is shown in Algorithm 1.



(a) (b)

Fig. 11. (a) The point NNBs in region 16, (b) The region 14 and 16 NNBs

Algorithm 1. Region NNB

1: **for** each point in the region **do**
2: Calculate NNB scope of this point$\{d_1, d_2, \ldots, d_n\}$
3: **for** each dimension d_i **do**
4: $\text{MAX}d_i \leftarrow \max(\text{MAX}d_i, d_i)$
5: $\text{MIN}d_i \leftarrow \min(\text{MIN}d_i, d_i)$
6: **end for**
7: **end for**
8: **return** $\{(\text{MAX}d_1, \text{MAX}d_2 \ldots), (\text{MIN}d_1, \text{MIN}d_i \ldots)\}$

The NNB calculation consists of two main steps in each iteration, as shown in Algorithm 1. The first step is to find the maximum boundary of the region NNB by comparing the maximum boundary of the region NNB with each dimension of point in the region (line 4). The second step is to find the minimum boundary of region NNB by comparing the minimum boundary of the region NNB with each dimension of point in the region (line 5).

4.3. *Parallel and distributed computing*

As described in the process of algorithm, grouping the points of each region by NNB on storage nodes is independent from each other. After grouping data, the tasks of computing the NN pair in each group of computation nodes are independent from each other too. So, the DHC algorithm can be accelerated by using parallel and disturbed computing frameworks. The DHC algorithm is shown in Algorithm 2.

In Algorithm 2, the distributed storage and computation technology is applied to group data of each leaf node in a kd-tree (line 4) and the distributed computation is employed to produce the mutual nearest-neighbor pairs of each group (line 5). The DHC algorithm can be significantly accelerated by using distributed storage and computation.

Algorithm 2. The DHC Algorithm

1: Initialize the items of the input dataset as original clusters
2: **repeat**
3: Store original clusters in kd-trees on distributed storage nodes
4: Group the points in the NNB of each region on storage nodes
5: Use distributed computation node to calculate LMN pairs of each region
6: Get all GMN pairs from LMN pairs
7: Merge clusters by GMN pairs to new clusters
8: **until** No new clusters are produced

5. Evaluation

In this section, the DHC algorithm's space complexity and time complexity are analyzed at first. Then, the important kd-tree parameter, namely the split number threshold T_{split} is discussed in the adjustment and influence of T_{split} to the DHC algorithm time cost.

5.1. *Space and time complexity analysis*

The space complexity of a quad-tree or kd-tree is $O(n)$, where n is the items number of a dataset. The space complexity of searching MNs is $O(m^2)$, where m is the maximum number of points in a leaf node. The all of space complexity is $O(n + m^2)$. In the DHC algorithm, data are stored on s distributed storage-service nodes. Within each node, a quad-tree or kd-tree is also used for storing data. So, the space complexity of DHC in each storage-service node is $O(\frac{n}{s} + m^2)$. Besides, storage control server records the data scope managed by each node, the space complexity of which is $O(s)$.

The time complexity of storing all points in a quad-tree or kd-tree is $O(n \log \frac{n}{m})$. The rule of the leaf nodes partitions in a quad-tree or kd-tree is that the partitioned regions cannot overlap with each other. Thus, the time complexity of the grouping data is $O(\frac{n}{m} \log \frac{n}{m})$. In the searching for MN pairs for each region, the time complexity is $O(m^2)$. The total time complexity of all regions is $O(\frac{n}{m} m^2) = O(\text{nm})$. The time complexity of each iteration is $O((n + \frac{n}{m}) \log \frac{n}{m} + \text{nm})$. In practice, m is much smaller than n, the time complexity is $O(n \log n)$ of each iteration. In DHC, the number of iterations is about $\log n$, so the time complexity of the DHC algorithm is $O(n \log^2 n)$. Based on the DHC algorithm time complexity analysis, if the algorithm uses s computing nodes to accelerate the computing by parallel and distributed computing framework, the amount of data onto each storage-service node is $\frac{n}{s}$. In this case, data are stored directly into corresponding nodes under control of storage server, so the time complexity of inserting points is $O(\frac{n}{s} \log \frac{n}{\text{ms}})$ and the time complexity of grouping data for all groups is $O(\frac{n}{\text{ms}} m^2) = O(\frac{\text{nm}}{s})$. In the iterating step, the total time complexity of DHC is $O((\frac{n}{s} + \frac{n}{\text{ms}}) \log \frac{n}{ms} + \frac{\text{nm}}{s})$ in each iteration.

5.2. *Adjustment and influence of threshold*

The DHC algorithm has only one parameter T_{split} which is the split threshold of leaf node and need to be adjusted. The maximum number of points in leaf nodes is about T_{split}. The average number of points in leaf nodes, m is linear correlation with $T_{\text{split}}, T_{\text{split}} = c \times m$, where c denotes a constant coefficient. After the analysis of the DHC algorithm time complexity in each iteration, the problem of T_{split} adjustment is

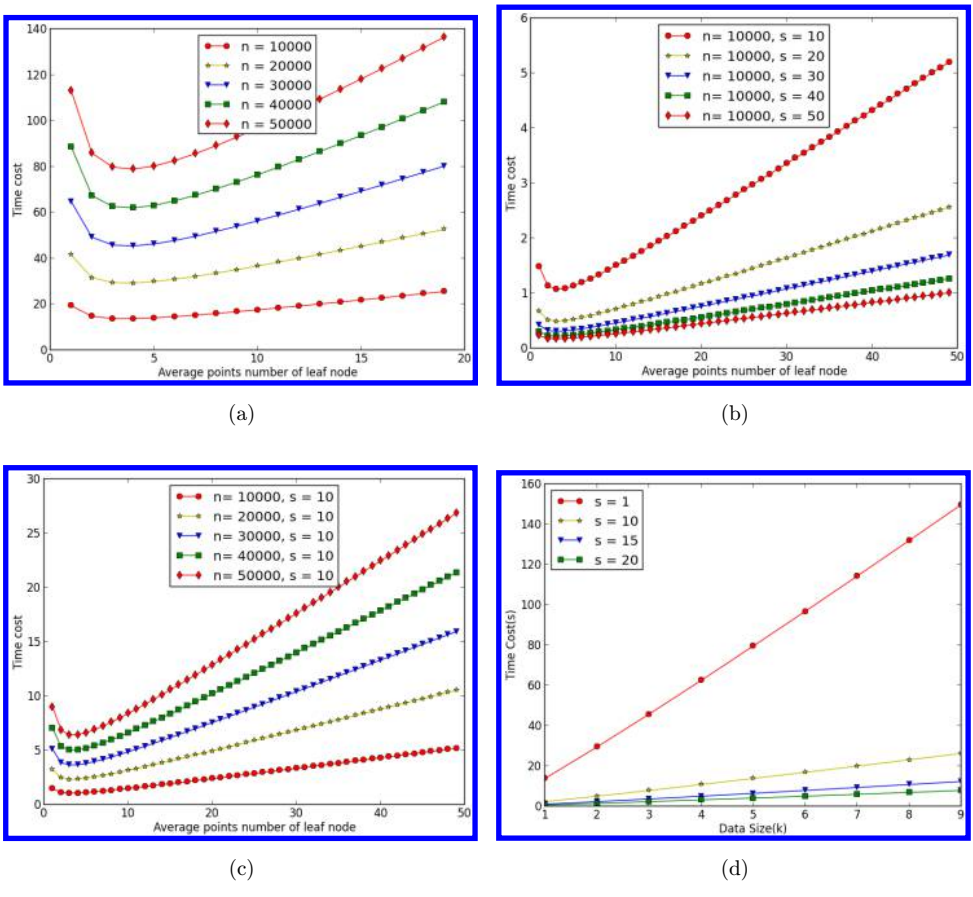


Fig. 12. The time cost of our algorithm DHC using (a) different data size n and average points number of leaf node m, (b) different s computing nodes and average points number of leaf node m, (c) different data size n and average points number of leaf node m when computing nodes number s is 10, (d) different data size n and computing nodes number s.

transferred to an optimization problem as

$$\arg\min_{m} \left\{ \left(n + \frac{n}{m} \right) \log \frac{n}{m} + \mathrm{nm} \right\}. \tag{10}$$

Let $\phi(m) = (n + \frac{n}{m}) \log \frac{n}{m} + \mathrm{nm}$, where $\phi(m)$ is the time complexity function of DHC. The second derivative of $\phi(m)$ is always positive. We can get the optimal solution by solving the following equation:

$$m - \sqrt{\log \frac{n}{m} + m + 1} = 0\,. \tag{11}$$

As shown in Fig. 12(a), the optimal solution denoted as m^* exists in different data size. And the accurate optimal solutions are shown in Table 2.

Table 2. The optimal solutions of m and their corresponding time cost using our algorithm DHC under varying data sizes.

Data size	m^*	Time cost
10 k	3.53	13.73
20 k	3.64	29.23
30 k	3.70	45.4
40 k	3.74	61.99
50 k	3.77	78.9
60 k	3.80	96.07
70 k	3.82	113.44
80 k	3.84	130.99
90 k	3.86	148.7

The DHC algorithm can use s computing nodes to accelerate the computing. From the analysis of the DHC time complexity, the problem of T_{split} adjustment is transferred to an optimization problem, which is formulated as

$$\arg\min_{m} \left\{ \left(\frac{n}{s} + \frac{n}{\text{ms}} \right) \log \frac{n}{\text{ms}} + \frac{\text{nm}}{s} \right\}. \tag{12}$$

According to the time complexity analysis, a large number of points in leaf nodes of a kd-tree can reduce the insertion and data grouping steps' time cost, but it enlarges the MNs searching time cost. In DHC, using more storage and computation node can reduce the time cost of MNs searching. Thus, the optimal solution m^* is calculated by solving the following equation:

$$m - \sqrt{\log \frac{n}{\text{ms}} + m + 1} = 0. \tag{13}$$

As shown in Fig. 12(b), we can get the best time cost in different s computing nodes by using the optimal solution m^*. The accurate optimal solutions m^* are shown in Table 3. From Fig. 12(c), the time cost increases slowly when we use more computing nodes, where $s = 10$. And the accurate optimal solutions m^* are shown in Table 4. When we employ more computing nodes for the DHC algorithm, the performance will be accelerated, as shown in Fig. 12(d).

According to the above analysis of the DHC algorithm, the DHC algorithm performance is under the influence of T_{split}. The performance of the DHC algorithm is improved efficiently by using more computing nodes.

5.3. *Synthetic data experiment*

An experimental study on synthetic datasets is presented for the DHC algorithm. The experiment is conducted on a server which has 2.5 GHz Intel Xeon CPU E5-2640

Table 3. The optimal solutions of m and their corresponding time cost using our algorithm DHC under varying computing nodes number s.

Data size	s	m^*	Time cost
10 k	10	2.45	1.09
10 k	20	2.32	0.5
10 k	30	2.24	0.32
10 k	40	2.18	0.23
10 k	50	2.14	0.18
10 k	60	2.1	0.14
10 k	70	2.07	0.12
10 k	80	2.04	0.1
10 k	90	2.02	0.09

Table 4. The optimal solutions of m and their corresponding time cost using our algorithm DHC under varying data sizes when computing nodes s is 10.

Data size	s	m^*	Time cost
10 k	10	2.45	1.09
20 k	10	2.58	2.36
30 k	10	2.65	3.7
40 k	10	2.7	5.08
50 k	10	2.74	6.49
60 k	10	2.77	7.93
70 k	10	2.8	9.39
80 k	10	2.82	10.87
90 k	10	2.84	12.37

with 24 cores. In the DHC algorithm, we run 10 nodes for distributed storage and computation. To validate the efficiency of the DHC algorithm for large datasets, we generate a set of synthetic datasets that are based on Gaussian distributions of different data sizes. An example of synthetic datasets is illustrated in Fig. 13.

AHC[1] is a classical algorithm to provide a hierarchical representation for the purpose of data analysis and still has been widely used in recent researches.[28–32] For example, a prototype-based AHC method is used for uncertain data clustering[28]; based on the dominant sets algorithm and cluster merging, a parameter independent clustering algorithm is presented[29]; a new linkage method NC-link is proposed for large-scale data[30]; a frequency based Dynamic Automatic Agglomerative Clustering is developed and presented over the two-dimensional datasets[31]; a hierarchy based on a group of centroids is built to make AHC efficiently.[32] Based on the novel concept of MN,[5] NNC and MLHC reduce the computation complexity of the AHC[1] algorithm. MN is also one important method to reduce the computation complexity of the hierarchical clustering algorithm in our DHC algorithm. Considering this, we compare DHC with AHC,[1], NNC[4] and MLHC[5] algorithm in the experiments.

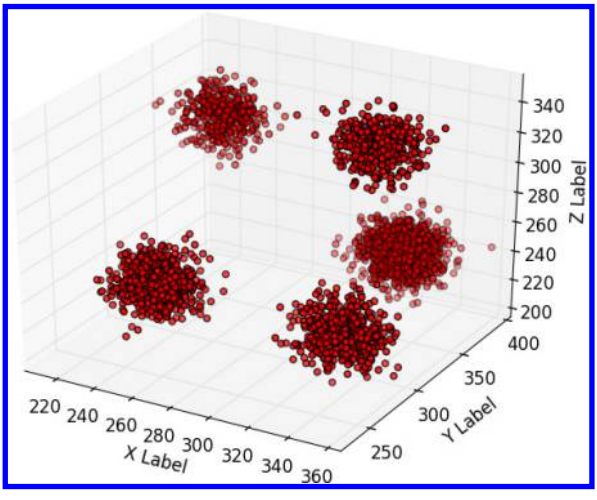


Fig. 13. An exampled synthetic dataset generated by Gaussian distribution.

Based on the experiments, we investigate the results of AHC, NNC and MLHC. We compare the time cost between algorithms under different sizes, datasets and analyzed the results and performances of DHC. The performance of algorithms is compared, as shown in Fig. 14.

The computation complexity of AHC is $O(n^3)$, the computation complexity of NNC is $O(n^2)$, the computation complexity of MLHC is $O(n^2 \log n)$ and the computation complexity of DHC is $O(n \log^2 n)$. From the figure, we can see that the time cost of the AHC algorithm is unacceptable when synthetic datasets grow up more than 9000 items. And the DHC algorithm has low time cost than NNC and MLHC

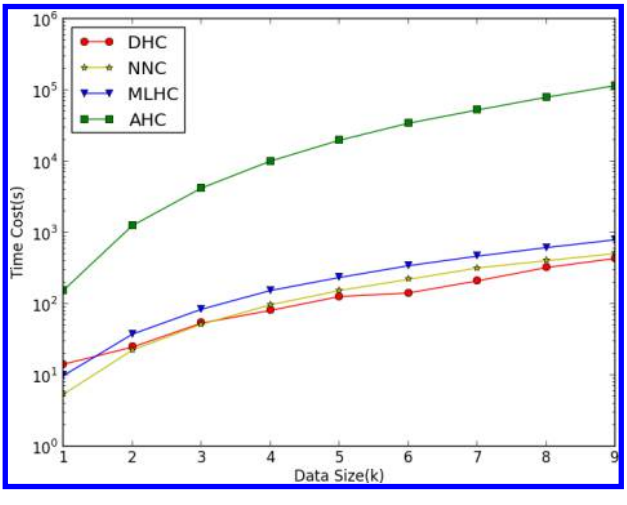


Fig. 14. The time cost of DHC, NNC, MLHC and AHC under varying data sizes.

Table 5. The time cost of DHC, NNC, MLHC and AHC under varying data items.

Data items	DHC	NNC	MLHC	AHC
1 k	14.27 s	5.4 s	9.8	155.6 s
2 k	24.81 s	22.6 s	37.7	1263.2 s
3 k	53.89 s	51.7 s	84.3	4234.1 s
4 k	81.19 s	97.3 s	154.4	10133.4 s
5 k	127.14 s	154.3 s	235.4	19884.6 s
6 k	142.64 s	221.8 s	344.5	34405.2 s
7 k	211.37 s	319.8 s	469.5	52888.1 s
8 k	324.41 s	406.1 s	618.3	79824.8 s
9 k	436.54 s	509.2 s	796.4	116178.1 s

algorithms when datasets increase. The experiments show that our DHC algorithm has fine performance. Due to their high-time complexity, the runtime of these algorithms may be weeks and even months when they deal with massive datasets. So, we just use small datasets in the experiments. In practice, the datasets often exceed the memory capacity of a computer and our DHC algorithm can deal with the massive dataset by distributed storage and computing.

The detailed time cost of DHC, NNC and MLHC are shown in Table 5. From the table, the time cost of DHC is greater than that of NNC at beginning. When the number of data items (greater than 4 k) increases, the performance of DHC is clearly accelerated. But when the size of the data items is less than 4 k, NNC and MLHC are efficient than DHC. We know that the computation complexity of DHC is $O(n \log^2 n)$ and we just use small datasets which do not exceed the main memory of a normal computer in the experiments. And the startup and the I/O operation of DHC need a large amount of time for distributed storage and computing. So, the speed acceleration of DHC is not obvious in these experiments. This is because the startup and the I/O operation need a large amount of time. And all F-scores of algorithms are one, indicating the four algorithms are the best solution to the synthetic datasets.

5.4. *Real-world datasets*

UCI data repository[37] Iris and UKMD are used in our experiments. The Iris flower dataset is a multivariate dataset and is an example of discriminant analysis.[38] Sometimes, it is called Anderson's Iris dataset because the data to quantify the morphologic variation of Iris flowers is collected by Edgar Anderson.[39] The Iris dataset consists of 50 samples and in each of these there are three species of Iris. Four features, including the length and the width of the sepals and petals, are measured from the samples. The UKMD is a dataset about the students' knowledge status.[40] The UKMD dataset includes 403 instances. Each instance has five features (STG, SCG, STR, LPR and PEG).

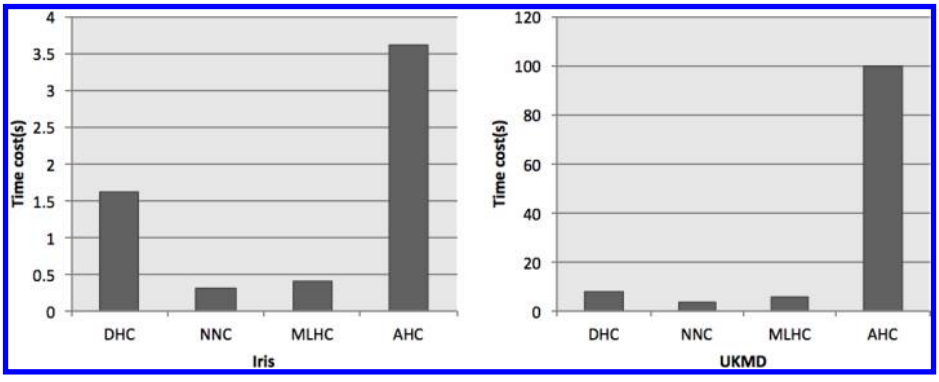


Fig. 15. The time cost of DHC, NNC, MLHC and AHC on real datasets Iris and UKMD.

From Fig. 15, DHC has a lower time cost than AHC,[1] but a higher time cost than NNC[4] and MLHC.[5] This is because Iris and UKMD are small datasets, and the startup and the I/O operation of DHC need an amount of time. All those algorithm employs Ward's method[41] as linkage function which satisfies the cluster aggregate inequality.[33,5] Based on the cluster aggregate inequality, the F-scores achieved by the four algorithms on Iris and UKMD datasets are the same, that on Iris the F-score is 89.5% and on UKMD the F-score is 53.0%.

For document clustering, LOG and Reuter[42] are common benchmark text datasets. Vector space model is used to represent documents as term vectors. The stop words and unnecessary tags and headers are removed from document datasets in preprocessing. LOG and REUTER are lager text datasets that LOG has 1367 documents and 400 features, REUTER has 2787 documents and 1000 features.[42] Figure 16 shows that the time cost of DHC is lower than NNC[4] and MLHC.[5] The DHC algorithm is accelerated obviously on larger datasets, LOG and Reuter. And the F-scores of them are 66.38% of LOG and 73.46% of REUTER.

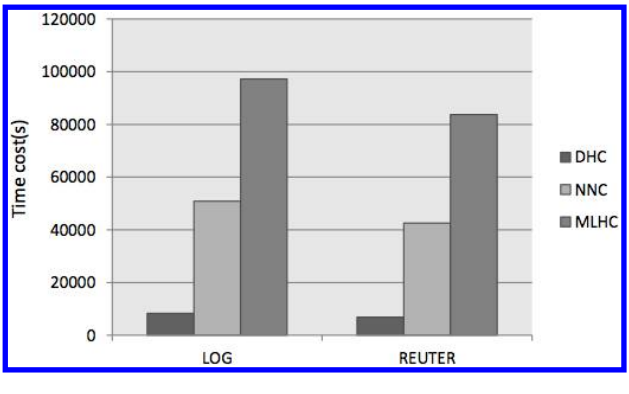


Fig. 16. The time cost of DHC, NNC and MLHC on real datasets LOG and REUTER.

According to the above real-world dataset experiments of the DHC algorithm, the DHC algorithm has a better performance than AHC, NNC and MLHC when dataset size is big. And the DHC algorithm is flexible to improved performance by using more computing nodes.

6. Conclusion

Hierarchy clustering is a classic data mining method, which widely used in analyzing physical or abstract objects and classifying them by similarity computing. However, the standard AHC algorithm has a high time complexity of $O(n^3)$ and a high space complexity of $O(n^2)$, which makes the computing too slow, even for medium-size datasets. This paper focuses on designing a distributed storage and computation scheme to deal with large datasets, and proposes a DHC algorithm. Based on NNB, DHC stores data onto kd-trees distributed among storage nodes and divides the data into subgroups in which the NN search is independent of each other. In this way, DHC can greatly reduce the time complexity by searching the NN in partitioned group. To further accelerate the computing speed of DHC, distributed computation framework is constructed to parallelize the DHC algorithm. We have conducted numerous experiments of synthetic and real-world datasets to evaluate the effectiveness of our DHC in accelerating computing speed. Experimental results show that the DHC algorithm outperforms the benchmarking algorithms in reducing time complexity of the loss of clustering accuracy.

Acknowledgments

This work is supported by the National Science Foundation of China under Grant No. 61272420 and the Jiangsu Government Scholarship for Overseas Studies.

References

1. P. Tan, M. Steinbach and V. Kumar, *Introduction to Data Mining* (Addison-Wesley Longman, Inc., 2005), pp. 516–518.
2. R. Sibson, Slink: An optimally efficient algorithm for the single-link cluster method, *Computer Journal* **16**(1) (1973) 30–34.
3. D. Defays, An efficient algorithm for a complete link method, *Computer Journal* **20**(4) (1977) 364–366.
4. F. Murtagh, A survey of recent advances in hierarchical clustering algorithms, *Comput. J.* **26**(4) (1983) 354–359.
5. C. Ding and X. He, Knowledge Discovery in Databases: PKDD 2005, *European Conference on Principles and Practice of Knowledge Discovery in Databases*, Porto, Portugal, October 3–7, 2005, Proceedings DBLP, pp. 71–83.
6. E. Chávez, M. Graff, G. Navarro and E. S. Téllez, Near neighbor searching with k nearest references, *Information Systems* **51**(C) (2015) 43–61.

7. C. M. Toledo, R. J. Barrientos and A. I. Ávila, Similarity (range and knn) queries processing on an intel xeon phi coprocessor, *Cluster Computing* **19**(1) (2016) 1–15.
8. W. Zhang, G. Zhang, Y. Wang, Z. Zhu and T. Li, NNB: An efficient nearest neighbor search method for hierarchical clustering on large datasets, *IEEE Semantic Computing (ICSC)*, March 2015, pp. 405–412.
9. R. A. Finkel and J. L. Bentley, Quad trees a data structure for retrieval on composite keys, *Acta Inform.* **4**(1) (1974) 1–9.
10. M. D. Berg, M. V. Kreveld, M. Overmars and O. Schwarzkopf, Computational geometry, *Comput. Geom.* (2000) 291–306.
11. H. Samet and R. E Webber, Storing a collection of polygons using quadtrees, *ACM TOG* **4**(3) (1985) 182–222.
12. J. L. Bentley, Multidimensional binary search trees used for associative searching, *Commun. ACM* **18**(9) (1975) 509–517.
13. I. Wald and V. Havran, On building fast kd-trees for ray tracing, and on doing that in o (n log n), *IEEE Symp. Interactive Ray Tracking*, 2006, 61–69.
14. W. Barreiros, G. Teodoro, T. Kurc, J. Kong, A. C. M. A. Melo and J. Saltz, Parallel and efficient sensitivity analysis of microscopy image segmentation workflows in hybrid systems, *IEEE Int. Conf. Clust. Comput.* **25** (2017).
15. Y. Qu, L. Lin, F. Shen, C. Lu, Y. Wu, Y. Xie and D. Tao, Joint hierarchical category structure learning and large-scale image classification, *IEEE Trans. Image Process.* **26** (2017) 4331–4346.
16. M. Wigness, B. A. Draper and J. R. Beveridge, Efficient label collection for image datasets via hierarchical clustering, *Int. J. Comput. Vis.* **126**(11) (2017) 1–27.
17. M. Gagolewski, M. Bartoszuk and A. Cena, Genie: A new, fast, and outlier-resistant hierarchical clustering algorithm, *Inf. Sci.* **363** (2016) 8–23.
18. S. Xu, Z. Han, X. Fu, J. Zhou and M. Chen, GPU-based fluid motion estimation using energy constraint, *J. Circuits Syst. Comput.* **26**(2) (2016) 400–433.
19. T. Zhang, R. Ramakrishnan and M. Livny, Birch: A new data clustering algorithm and its applications, *Data Min. Knowl. Discov.* **1**(2) (1997) 141–182.
20. M. A. Abbas and A. A. Shoukry, Cmune: A clustering using mutual nearest neighbors algorithm, *Information Science, Signal Processing and their Applications (ISSPA)*, Montreal, QC, Canada, 2012, pp. 1192–1197.
21. S. Guha, R. Rastogi and K. Shim, Cure: An efficient clustering algorithm for large databases, *SIGMOD Rec.* **26**(1) (1998) 73–84.
22. G. Karypis, E. Han and V. Kumar, Chameleon: Hierarchical clustering using dynamic modeling, *Computer* **32**(8) (1999) 68–75.
23. A. McCallum, K. Nigam and L. H. Ungar, Efficient clustering of high-dimensional data sets with application to reference matching, *Proc. 6th Int. Conf. KDD ACM*, Boston, Massachusetts, 2000, pp. 169–178.
24. C. Jin, Z. Chen, W. Hendrix, A. Agrawal and A. Choudhary, Incremental, distributed single-linkage hierarchical clustering algorithm using mapreduce, *Proc. Symp. High Perform. Comput. (SHPC)*, April (2015), pp. 83–92.
25. S. Moon, J. G. Lee, M. Kang, M. Choy and J. W. Lee, Parallel community detection on large graphs with mapreduce and graphchi, *Data Knowl. Eng.* **104** (2016) 17–31.
26. T. Sun, C. Shu, F. Li, H. Yu, L. Ma and Y. Fang, An efficient hierarchical clustering method for large datasets with map-reduce, *International Conference on Parallel and Distributed Computing, Applications and Technologies* (IEEE, 2010), pp. 494–499.
27. W. H. Day and H. Edelsbrunner, Efficient algorithms for agglomerative hierarchical clustering methods, *J. Classif.* **1**(1) (1984) 7–24.

28. F. Gullo, G. Ponti, A. Tagarelli and S. Greco, An information-theoretic approach to hierarchical clustering of uncertain data, *Inform. Sci.* **402** (2017) 199–215.
29. J. Hou and W. Liu, Parameter independent clustering based on dominant sets and cluster merging, *Inform. Sci.* **405** (2017) 1–17.
30. Y. Jeon, J. Yoo, J. Lee and S. Yoon, Nc-link: A new linkage method for efficient hierarchical clustering of large-scale data, *IEEE Access* **5**(99) (2017) 5594–5608.
31. M. Madheswaran and K. S. Sreedhar, An improved frequency based agglomerative clustering algorithm for detecting distinct clusters on two dimensional dataset, *Journal of Engineering and Technology Research* **9**(4) (2017) 30–41.
32. A. Bouguettaya, Q. Yu, X. Liu, X. Zhou and A. Song, Efficient agglomerative hierarchical clustering, *Expert Syst. Appl.* **42**(5) (2015) 2785–2797.
33. B. Mirkin, Mathematical classification and clustering: From how to what and why, Springer (1998) 139–158.
34. G. N. Lance and W. T. Williams, A general theory of classificatory sorting strategies II. clustering systems, *Comput. J.* **10** (1967) 271–277.
35. J. Zhou, M. Yin, Z. Li, K. Cao, J. Yan, T. Wei, M. Chen and X. Fu, Fault-tolerant task scheduling for mixed-criticality real-time systems, *J. Circuits Syst. Comput.* **26**(1) (2017) 1750016-1–1750016-17.
36. R. Bellman, Dynamic programming, *Dynam. Program. Science* **153**(3731) (1966) 34–37.
37. C. Blake and C. J. Merz, UCI repository of machine learning databases, ftp://ftp.ics.uci.edu/pub/machine-learning-databases (1998).
38. R. A. Fisher, The use of multiple measurements in taxonomic problems, *Annals of Eugenics* **7**(2) (1936) 179–188.
39. E. Anderson, The species problem in iris, *Ann. Missouri Bota. Gard.* **23**(3) (1936) 457–509.
40. H. T. Kahraman, S. Sagiroglu and I. Colak, The development of intuitive knowledge classifier and the modeling of domain dependent data, *Knowl.-Based Syst.* **37**(2) (2013) 283–295.
41. J. W. Jr., Hierarchical grouping to optimize an objective function, *J. Am. Stat. Assoc.* **58**(301) (1963) 236–244.
42. T. Li and C. Ding, Weighted consensus clustering, *Proc. SIAM Int. Conf. Dataming (SDM)* (2008), pp. 798–809.