DDCM ： A Decentralized Density Clustering and Its Results Gathering Approach

Lida Zou (zoulida@sdufe.edu.cn)
Shandong University of Finance and Economics https://orcid.org/0000-0001-8511-9243

Research Article

Keywords: distributed clustering, decentralization, results gathering, density clustering, content addressable network

Posted Date: August 1st, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1869186/v1

DDCM：A Decentralized Density Clustering and Its Results Gathering Approach

Lida Zou [1,2,*]

1. School of Computer Science and Technology, Shandong University of Finance and Economics, Jinan 250014, China
2. Key Laboratory of Machine Learning and Financial Data Mining in Universities of Shandong, Jinan 250014, China
**. Corresponding author email: zoulida@sdufe.edu.cn*

Abstract: Distributed clustering is an important way to solve large-scale data mining problem. At present, there still exist some problems for distributed clustering, such as performance bottleneck of master node and network congestion caused by global broadcast. In the paper, we propose a decentralized clustering method based on density clustering and content-addressable network technique. It could generate a cluster just depending on the surrounding several nodes, which realizes load balancing and has excellent scalability. An optimization way of gathering clustering results in different application scenarios is addressed as well. Our extensive experiments show that the proposed approach performs 3 times better than benchmark algorithm in time efficiency and has a stable expansion ratio at about 0.6 for large-scale data.

Keywords: distributed clustering; decentralization; results gathering; density clustering; content addressable network

1 Introduction

With the rapid development of information technology, massive data and information from human's life and production can be gathered and stored. There are increasing amounts of data stored into database, which makes cluster analysis of extremely large datasets become particularly important. The current clustering algorithms [1]–[3] present issues of scalability and efficiency for extremely large-scale data. Cluster analysis on extremely large-scale data is a critical topic in the data mining research. Distributed clustering [4], [5] is a most effective way to improve clustering efficiency.

Distributed clustering abstracts classification pattern from large-scale data in distributed computing environment. Its common idea is as follows [6]. First local clustering is done on local nodes. Second local clustering results or parameters are sent to master node by local nodes, and master node clusters these received data globally to get global clustering models. Last master node sends these models to each node, and each node updates their clusters according to the global models.

All nodes are required to gather information and broadcast information during distributed clustering. When the number of distributed nodes is large, the amount of network traffic for master node would increase massively. In the case distributed clustering takes more time than centralized clustering.

A decentralized architecture with high autonomy in each node could effectively alleviate the load of master node in distributed clustering. Decentralized system no longer needs master node, and each node finishes computing tasks in an equal and cooperative way. The decentralized idea is introduced into clustering of mass data, which could balance the load of each node and improve computing efficiency. The peer-to-peer network is a classic, decentralized architecture. Some algorithms such as DBDC[7], k-DmeansWM[8] try to utilize peer-to-peer network to finish distributed clustering. However, clustering computation needs global information, which was realized via wide information broadcast or setting a master node to share information. Thus clustering based on peer-to-peer network has the issues of load imbalance or network overload as well.

In the algorithm called DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [9], the generation of a cluster just depends on local information and no longer needs global parameters, which could suitably be broadened to decentralized environments. Content Addressable Network (CAN)[10] is a decentralized data distribution method. It balances multidimensional datasets across the nodes and similar data are distributed nearby. In the paper we combine the above two techniques and propose a Decentralized Density Clustering Method (DDCM). Its main idea is as follows. First clustering objects are distributed among CAN spaces according to their attributes. Second each node finishes clustering cooperatively using DBSCAN algorithm in CAN space. The above two steps will be discussed in Section 3 and Section 4 in details.

DDCM follows the decentralized principle. During the procedure of clustering, each node does not rely on global information and they can form a cluster using several adjacent nodes. DDCM eliminates performance bottlenecks and is no longer limited to the number of nodes. The number of clustering objects can increase linearly as the number of nodes rises.

When distributed clustering of datasets is completed, there is also the issue of gathering clustering results. The demands of gathering results are varied. In anomaly detection, a few abnormal samples are needed, while in classification the generated clusters are collected. In a distributed environment, results gathering is a key part with heavy network load. There still is great optimized space for different demands of gathering results. In the paper we give the methods of gathering results in different application scenarios according to the characteristics of DDCM, which will be introduced in Section 5.

The main contributions of our work are as follows.

1) A decentralized clustering algorithm is proposed, which realizes load balancing, reduces network load and has good scalability.

2) We analyze application scenarios of gathering results in distributed clustering, and design different optimized schemes. It could further enhance the application efficiency of distributed clustering.

2 Related Work

Distributed clustering is a necessary path to improve time efficiency of clustering algorithm for very large datasets [4]. There has been a lot of research on distributed clustering for academia, which fall into three kinds.

The first one is distributed clustering based on MapReduce [11]–[15]. Its main idea is to adaptively improve classical clustering algorithms using MapReduce, and complete the calculation in parallel. However, the framework of MapReduce is stylized. When doing clustering calculation, it follows a pattern that executes the operation of Map then Reduce, which makes it hard to flexibly adapt data characteristics and clustering demands. The way is likely to develop problems of data skew and slow convergence.

The second one is distributed clustering with master node [6], [16]–[20][7], [21]. The approach sets a master node to manage nodes, gather and broadcast the required intermediate results or parameters [7]. Moreover, its accuracy rate is positively associated with the amount of data transferred. All above must come to being performance bottleneck of master node, especially for network traffic of master node.

The last one is distributed clustering based on P2P [8], [22]–[26]. It utilizes P2P protocol to organize nodes and distribute data. Each node is equivalent and there is no master node. However, when forming its clusters, all nodes are needed to exchange information globally and there will be heavy network loads. For example, each node broadcasts its own information and gathers information of all the other nodes after it locally clusters [8], which generates significant traffic. Li et al. [26] adopt tree structure to organize nodes and each node sends information to root node. In fact root node becomes master node and performance bottleneck is introduced.

Given the above, distributed clustering still has opportunity to optimize load balancing and network traffic. In the paper we propose DDCM algorithm, which can just use local information to finish clustering in decentralized network. It could effectively decrease network traffic and enhance time efficiency.

The Peer-to-Peer network is a common decentralized structure, and its mature techniques include CAN[10], BATON[27], Pastry[28], Chord[29]. CAN constructs overlay network based on multi-dimensional coordinate space, which is fit for multi-dimensional datasets in the decentralized environment.

3 Distribution of Clustering Objects over Decentralized Nodes

Before clustering computation, clustering objects are required to be distributed over computing nodes. In the paper we use CAN to organize the distribution, which is designed based on virtual space of d -dimensional Cartesian coordinates. The whole space is assigned to all the nodes and each node is responsible for maintaining a disjoint region. The nodes in CAN self-organize and form an overlay network representing the virtual space of coordinates. Each node needs to maintain the network address of neighbor nodes and generate its own routing table. Through routing table, the path between any two nodes in the space can be found.

Let dimensions of CAN space be equal to the attribute dimensions of clustering objects. Map each object to a unique point in CAN space according to its multi-dimensional attributes. All these points are called **data point**.

Assume that the attributes of data point is two-dimensional. Fig. 1 shows an example of mapping data points to nodes in two-dimensional CAN space. The dots denote data points. Blue one is normal point, while yellow one is abnormal. Each grid represents a computing node. It is observed that each node manages part of data points. Local node has both normal and abnormal points.

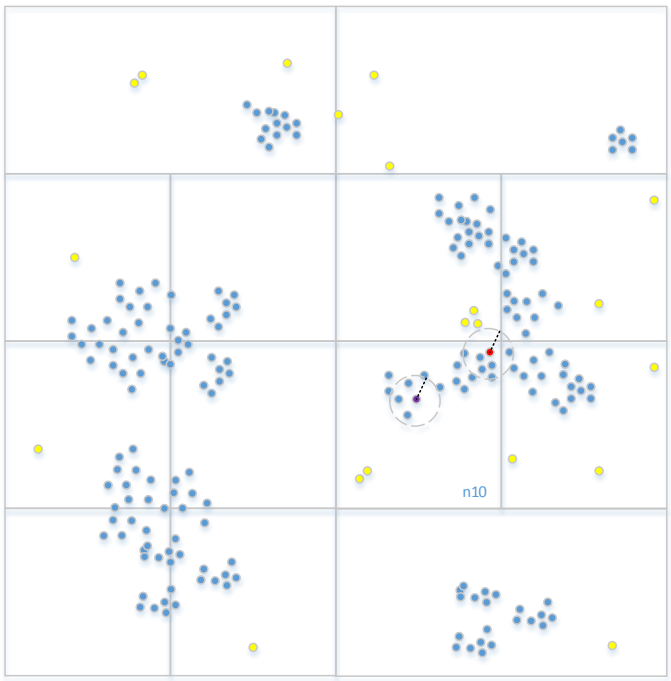

Fig. 1 Distribution of clustering objects on nodes in two-dimensional CAN protocol

4 Distributed Density Clustering in CAN Space

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a spatial clustering algorithm of high performance. Its basic idea is to put p and q into the same cluster if they are density-connected[9]. All the data points in the circular region of ε radius from data point p is called ε neighborhood of p, which is denoted as $N_\varepsilon(p)$. It is computed as $N_\varepsilon(p)=\{q \mid q \in D \wedge \mathrm{dist}(p,q) \le \varepsilon\}$, where D is the set of data points, $\mathrm{dist}(p,q)$ is the distance between p and q.

For a given parameter MinPts, if ε neighborhood of a data point p in D includes at least MinPts points, p is called core point; else it is called non-core point. For two points p and q in D, if q is in the ε neighborhood of p and p is a core point, q is directly density-reachable from p. If q is a core point and r in the ε neighborhood of q, r is density-reachable from p. Density-reachable nature can spread.

Since the relative location of clustering object in CAN space is unchanged, DBSCAN could be used in CAN space. It just computes the adjacent data points to cluster. Global information is not required and DBSCAN is suitable for decentralized distributed environment. There are still applicability of DDCM when using DBSCAN in decentralized environment. For the sake of description, we give the following definitions.

Definition 1 Node：The computing node organized based on CAN protocol is called *Node*. A node is also called local node. These nodes are peer-to-peer and completely decentralized. Each node has the same logic and function. Node is denoted as n_i, where i is the identifier.

We assume that the computing performance of each node is the same in order to simplify model. The computing task of each node, i.e., the number of data points, needs to be roughly equal.

Neighbor Node：For two given nodes A and B, responsible space of B is next to responsible space of A. Node A and B are neighbor node each other.

Adjacent Node：The space of node B is nearby the space of node A. Node B is an adjacent node of A. The set of adjacent nodes for A includes its neighbor node.

Definition 2 Data Point：The mapped points of clustering objects in CAN space according to their attributes. It can also be called point for short.

Local Data Point：In CAN protocol, a local node manages part of data points in D. These data points are called local data points of the local node. The node is called **hosting node** of these points.

DBSCAN first computes $N_\varepsilon(p)$ and then executes density clustering algorithm. In CAN, data points are distributed on local nodes. The computation of $N_\varepsilon(p)$ and density clustering algorithm is different from that in centralized DBSCAN. We next discuss computation method of $N_\varepsilon(p)$ and adaptive improvement of clustering algorithm.

4.1 $N_\varepsilon(p)$ Computation on Local Node

There are two kinds of data points during clustering. We call them **interior point** and **edge point**. For Node n10 in Fig. 2, the blue ones are interior points. Since ε neighborhood of interior point is in the space charged by local node, $N_\varepsilon(p)$ computation of interior node can be finished on local node. In Fig.2 yellow ones are edge points. Because ε neighborhood of edge point is partly on its neighbor node, $N_\varepsilon(p)$ computation of edge point needs the information of neighbor node.

For the yellow points of node n10 in Fig.2, their ε radius involves 3 neighbor nodes. $N_\varepsilon(p)$ computation needs n10 to exchange information with 3 nodes. In consideration of load balancing, the size of space charged by each node is different in CAN protocol. ε radius of edge point may have complicated distribution on its adjacent nodes. For example, Fig 3 shows a case of node distribution in two-dimensional CAN protocol. Each node has distinct size of space and their distribution is uncertain. $N_\varepsilon(p)$ computations of yellow points need assistance of shaded nodes in Fig.3. The task of computing $N_\varepsilon(p)$ requires complex distributed scheduling and much communication between nodes. If the number of edge points on local node is large, heavy network load would be caused.

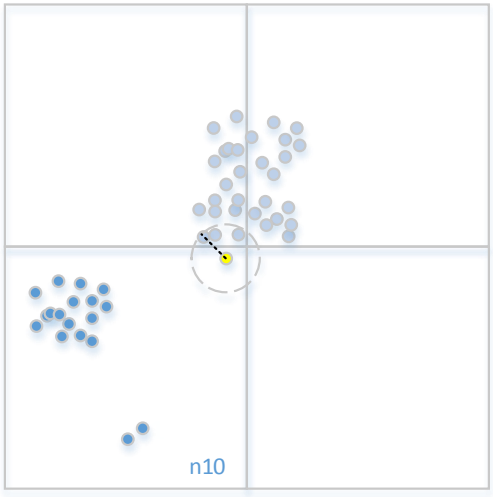

Fig. 2 Interior point and edge point on local node

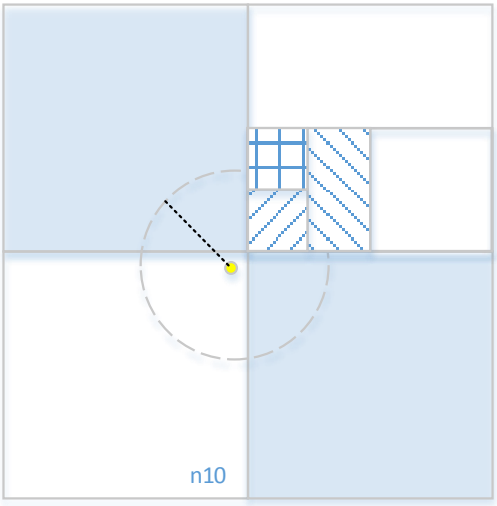

Fig. 3 A complex distribution case of ε radius around edge point

To solve the above problems, we design a distributed method of computing $N_\varepsilon(p)$, which could complete $N_\varepsilon(p)$ computation in CAN accurately and quickly. Its main idea is that information of adjacent nodes are collected in advance by local node and $N_\varepsilon(p)$ computation no longer appeals to for other nodes. We use two-dimensional space to explain the steps in the facilitation of illustration demonstration.

The steps of computing $N_\varepsilon(p)$ are as follows.

1) For a given node n10, an area with the width of ε is generated to surround its space before computing $N_\varepsilon(p)$, as shown the shaded area in Fig.4. Its steps are as follows. a) For each side of n10 space, expand a rectangle with width of ε; b) For angle of n10 space, expand a circular sector with ε radius and 90 degrees; c) Merge the above rectangles and circular sectors, and let G denote the surrounding area. The data points mapped into G are called **peripheral points** of n10.

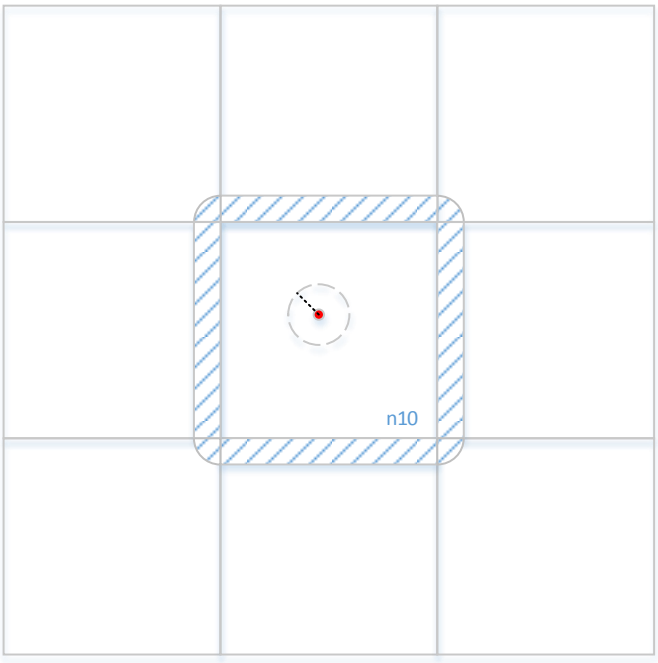

Fig. 4 The surrounding area with the width of ε

2) Randomly find a coordinate point in G. Node n10 finds the corresponding adjacent node according to the location of point. The adjacent node is assumed as n11.

3) Node n11 overlaps its own space with G and the overlapped part is denoted as g. Search the data points in g and send them to n10.

4) Node n10 adds these received points into its set of peripheral points in order to compute $N_\varepsilon(p)$.

5) Update G as $G \leftarrow G - g$

6) If $G = \varnothing$ stands, end; else repeat from steps 2) to steps 5).

7) Node n10 compute the value of $N_\varepsilon(p)$ based on peripheral points.

Steps 2) to 5) are executed repeatedly, and the data points in ε radius around edge point are all collected by n10. Therefore, $N_\varepsilon(p)$ computation of both interior point and edge point can just depend on local node to rapidly complete.

In distributed and rapid $N_\varepsilon(p)$ computation method we design, steps 2) to 5) are called ***collection process of peripheral points***.

4.2 Clustering Across Nodes

After finishing the computation of $N_\varepsilon(p)$, density clustering starts. It is done in two steps. First, local nodes separately execute DBSCAN algorithm on local data points and several clusters form. Second, if there are edge points, merge operation among nodes is needed. In Fig.2 we observed that yellow points should merge with the clusters of other nodes. The merged cluster is called as **cross-node cluster.** In the section, we discuss the merging method of cross-node clusters.

Local clustering uses classic DBSCAN algorithm. Interior points, edge points and peripheral points on local node participate in clustering, and many clusters generate. These clusters fall into **internal cluster** and **cross-node cluster**. In internal cluster, the points are interior points or edge points, and there are no peripheral points, just as cluster C0 in Fig.5. Internal cluster is a category without merging with clusters of other nodes.

In cross-node cluster, there are peripheral points, and data points of other nodes need to add into the cluster. Cross-node cluster is along the border of local node and there may be merging of two or more clusters. We conclude two cases of merging for cross-node clusters. The first one is that peripheral points in the cluster are core points of other nodes. This means that another node must have a cluster including the core point. In the case the clusters of the two adjacent nodes need merging. For cluster C1 in Fig.5, there is a yellow point, which is both core point and peripheral point. Through its density-reachable nature, C1 of n10 and C2 of n11 can be merged into a cluster. The second case is that peripheral points in the cluster are non-core points of other nodes. Non-core points cannot form a cluster on neighbor node. Merging clusters is not required and we just need to add the non-core points into the cluster. For example, orange point in Fig.5 is a peripheral point of n10 not a core point of n11, and it is added into C1 directly.

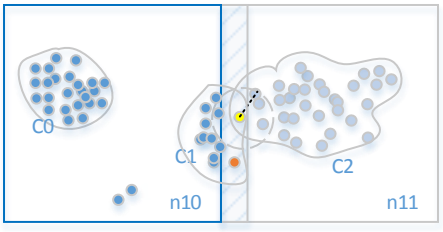

Fig. 5 Two cases of merging for cross-node clusters after local clustering

Merging algorithm of cross-node clusters requires two nodes' cooperation and it consists of two child processes. One is **Discovery Process**, which sends information of cross-node clusters on the node to neighbor node. The other is **Merge Process**, which executes merging after receiving merge request of discovery process.

We assume local node is denoted as n10. Algorithm 1 shows a concrete algorithm description of discovery process.

Input: Z , Z is the set of peripheral points on n10; $b_i.C_j \mid b_i \in Z$, b_i is peripheral point, i is identifier, $b_i.C_j$ means the cluster b_i belongs to after local clustering, j is identifier of clusters; $b_i.n_k \mid b \in Z$, $b_i.n_k$ is hosting node of b_i , k is identifier of node.

Algorithm: Discovery Process

FOREACH b_i in Z

$C1 \leftarrow b_i.C_j$ # Assign the cluster b belongs to to $C1$

IF there are only peripheral points in $C1$

$Z \leftarrow Z - C1$

CONTINUE

$D \leftarrow \{b_x \mid b_x \in C1 \cap Z, b_x.n = b_i.n\}$ # find all the peripheral points belonging to C1 and having the same hosting node with b_i on node n10.

Send the message of D classified as C1 to $b_i.n_k$

$Z \leftarrow Z - D$

Algorithm 1 Discovery Process

Each node always monitors the messages from neighbor nodes. When it receives the message of D classified as C1 from discovery process of neighbor node, merge process is called. Let n11 denote local node. Algorithm 2 gives the algorithm description of merge process.

Input： D ， $D \in C1$

Algorithm：Merge Process

$D' \leftarrow D$

FOREACH b_i in D'

IF b_i is non-core point

$b_i.C_j \leftarrow C1 | b_i$ in n11#update the identifier of cluster b_i belongs to on n11

ELSE # b_i is core point

$D' \leftarrow D' - A_i$, A_i is the set of density-reachable points for b_i on n11

IF $b_i.C_j == C1 | b_i$ in n11#the merging is finished

CONTINUE

ELSE

Execute Merging Function with parameters $C1$ and $b_i.C_j | b_i$ in n11#merge $C1$ and C_j

Algorithm 2 Merge Process

When merging cross-node clusters, the clusters each core point involves need to be merged. The first line in Algorithm 2 provides guarantee for the requirement, or merging operation can not be completed accurately. For example, for node n11 in Fig.6, C2 and C3 are not the same cluster after local clustering, but C2 and C3 become the same one through the link of C1 on node n10.

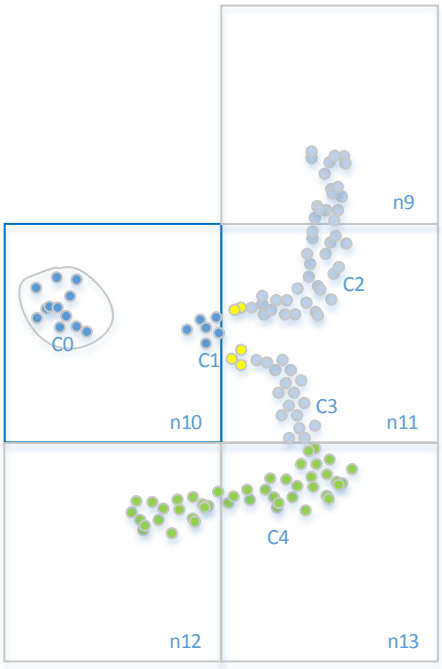

Fig. 6 Different clusters on local node become the same one after merge process

The metadata of cross-node cluster on local node consists of multiple tuples denoted as $< C_j, N_j >$ and each tuple corresponds to a cross-node cluster of the node, where j is the identifier of cross-node cluster, C_j is the set of its data points on the node, N_j is the set of hosting nodes the cross-node cluster is distributed on. When cross-node clusters of two nodes are merged, the information can quickly spread among the nodes the clusters involve through N_j.

5 Results Gathering

After clustering is completed, clustering results need gathering. The node receiving clustering results is called **integration node**. Gathering information on many nodes may cause massive network load. In distributed clustering scenario for large-scale data, there exist different demands for gathering results, and gathering efficiency can be optimized according to different application requirements. We next discuss three kinds of demands for gathering results.

The first one is to gather all the clustering results. It is similar to centralized clustering, which gathers information of all the clusters and outliers. For the demand, information on all the nodes should be gathered and the space of optimizing network load is limited.

The second demand is gathering information of only clusters. It could satisfy most application scenarios of clustering. For example, unsupervised learning just needs the information of clusters and their cluster members. For this demand, a **rapid gathering method** is required to reduce network load. We will discuss it in Section 5.1.

The third one is to gather the minimum or maximum value, such as finding the largest cluster or the most abnormal data point. The way of first gathering then comparing could cause severe performance bottleneck. If comparison operation is executed on local node, the bottleneck is removed. **A quick query approach** for the maximum and minimum value of clustering results is proposed in Section 5.2.

5.1 Clusters Collection

There are two ways of collecting clusters. The first way is that each node sends its own generated clusters to **integration node**, including both **internal cluster** and **cross-node cluster.** The method applies to the case of a few nodes. For a large number of nodes, **integration node** would have much network load in this way. The second way is to send information of hosting node in units of clusters. Internal cluster directly sends information, while cross-node cluster first sends information to a **relay node** and then reports to **integration node.** The method applies to the case of a few clusters. When the number of clusters is largely less than that of nodes, network load of **integration node** will decrease obviously.

The first method is simple and won't be covered here. For the second method, cross-node cluster chooses one node to report information and the node is called **primary node** of the cluster. After finishing the clustering of cross-node cluster, primary node reports to **integration node** the first time. Therefore, its primary node needs to timely determine the completion of all the hosting nodes for the cross-node cluster.

Primary node confirms with all the hosting nodes of cluster i according to N_j value in $< C_j, N_j >$. There still exists an issue when primary node determines whether merge process is completed. The merging of clusters is dynamic and the variable N_j will dynamically add nodes during merge processing.

5.2 Maximum or Minimum value Gathering

For some clustering applications, gathering all the results is not necessary. For example, anomaly detection just need the abnormal data and behavior analysis only requires some of the biggest clusters. Next we give a gathering method of maximum or minimum value for DDCM.

1) As shown in Fig.7, all the nodes build a competition tree in the division order of CAN and each node outputs maximum or minimum value to the competition tree.

2) The root node of competition tree is **integration node**, just as the red stroke in Fig. 7. The value of root node is maximum or minimum value.

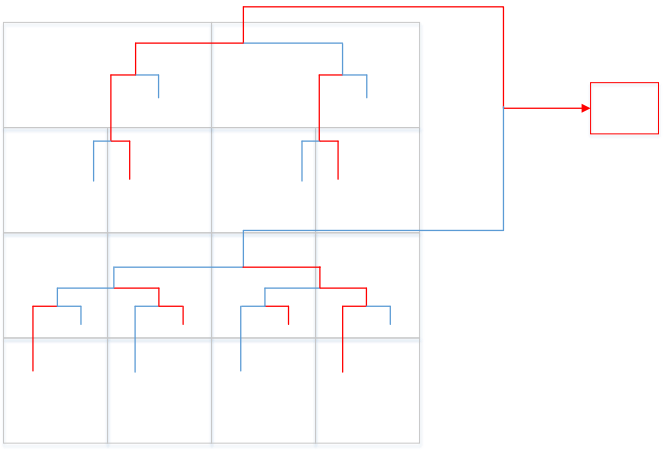

Fig. 7 Maximum or minimum value gathering based on competition tree

There are other gathering requirements in large-scale clustering scenario, such as determining whether two data points are in the same cluster. In most cases the rapid gathering method is designed according to the distribution characteristics of cross-node clusters.

6 Performance Analysis and Experimental Demonstration

In Section 6.1, we analyze the performance of DDCM, and its efficiency and scalability are validated experimentally in Section 6.2 to Section 6.4.

6.1 Performance Analysis

For clustering of local node in DDCM, time complexity of its two sub processes is $O(\frac{N}{M})$, where N is the number of data points, M is the number of nodes. In consideration of load balancing, the number of data points each node processed is about $\frac{N}{M}$. Therefore, time complexity of DBSCAN on local node is $O(\frac{N}{M}\log\frac{N}{M})$.

In clustering process, time complexity of network communication is $O(CK^2)$, where C is the number of cross-node clusters, K is the average number of hosting nodes for cross-node clusters. All the hosting nodes are informed for each time of merging. In extreme cases, merging may needs executing K times and the communication cost for generating a cross-node cluster is up to K^2 times.

DDCM algorithm completes clustering tasks based on decentralized architecture. The computation and communication cost of local nodes are low, and performance bottle is removed.

6.2 Experimental Setting

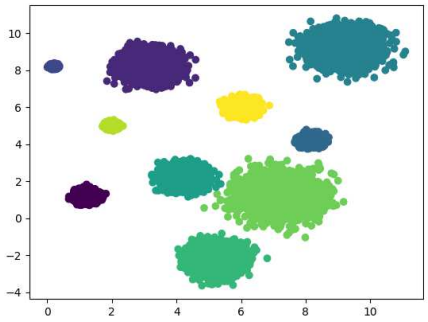

Fig. 8 Randomly generated clustering objects

We build 8 to 64 virtual nodes using OpenStack. Ubuntu operating system is installed on each node. Each node is assigned a processor, 2GB memory and 50GB disks. Clustering program is in Python. The data is randomly generated and their distribution is shown in Fig. 8. We assume the amount of data is no larger than 3 M.

Data should be sent to each node before clustering computation in distributed algorithm. When DDCM sends data to nodes of CAN, the nodes are divided according to the load balancing rule. In extreme cases, M times divisions may occur and it is time-consuming. In the experiments, we give two initialization methods for CAN. One is classical. First the average load is computed for each node. When the load of a node is larger than average load, the node is divided. The method is called DDCM in the experiment. The other is an improved method. Each node equally divides CAN space. After assigning the data points, repeatedly execute the following operations, i.e., if the difference between maximum load and minimum load is larger than a set

threshold, the node with the minimum load shares the loads of node with the maximum load. The method is called DDCM-A in the experiment.

As a comparison object, we choose MapReduce-Based Density Clustering (MBDC) [11][30] as a benchmark. To facilitate the experiments, we improve MBDC algorithm as follows. 1) Partition the vector space of data points and distribute data to the partitions. 2) In Map phase, execute local clustering for data points in each partition. 3) In Shuffle phase, send the data of neighboring partitions to the same node. 4) In Reduce phase, merge the clusters in different partitions and store them into HDFS.

6.3 Time Efficiency

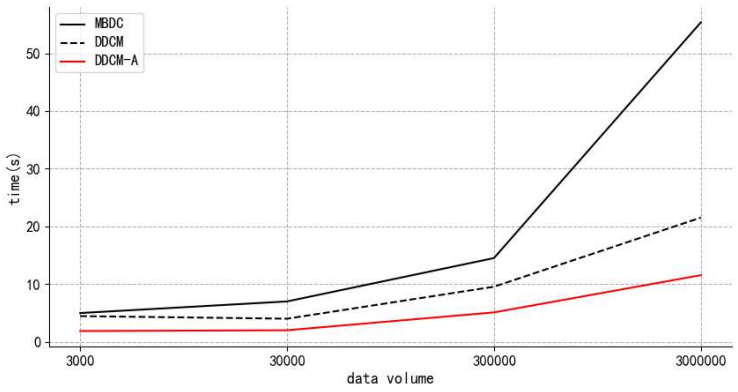

Fig. 9 Time efficiency in different amounts of data

In the subsection, we test time efficiency of DDCM. The amount of data is from 3K to 3M, and the number of nodes is assumed to be 32. As shown in Fig.9, DDCM-A and DDCM both perform better with the increase of data volume, but DDCM-A consumes less time than DDCM.

6.4 Scalability

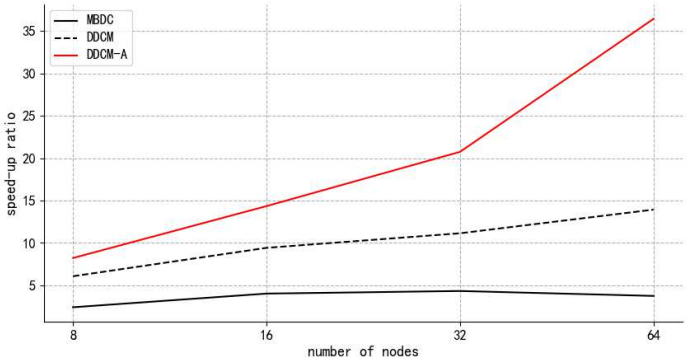

Fig. 10 Speedup ratio in different numbers of nodes

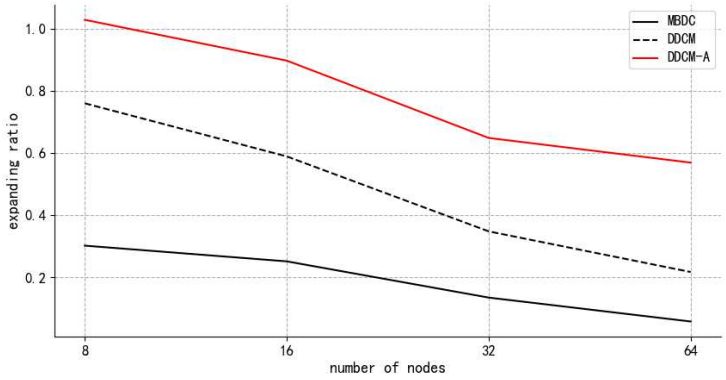

Fig. 11 Expanding ratio in different numbers of nodes

In the subsection, we validate speedup ratio and expanding ratio of DDCM. Speedup ratio is the time ratio of single node consumes and distributed execution consumes. Expanding ratio is the ratio between speedup ratio and the number of nodes. We assume that the amount of data is 3M and the number of nodes varies from 8 to 64. We observed that both DDCM-A and DDCM has stable speedup ratio and good expanding ratio when the number of nodes is large.

7 Conclusions

DDCM executes the distributed clustering tasks based on decentralized architecture. The generation of clusters just relies on several adjacent nodes, which does not set master node or need global information broadcast. The proposed method makes full use of large-scale distributed computation in a low network load. Experiments demonstrate that it removes performance bottleneck and has good scalability. Meantime, since each node has the same function and logic, its difficulties of distributed deployment and management are decreased. Therefore, DDCM could well adapt to the clustering demands of large-scale data. With the development of digital society, our method has potential applications and important values. In the future, we would focus on an automatic setting approach of the number of nodes according to the different data volumes, which could further reduce clustering cost.

Acknowledgements This work is supported by Shandong University of Finance and Economics, Jinan, China. At the same time, the author would like to sincerely thank the anonymous referees for their very valuable time.

Funding This work was supported by Science and Technology Plan for Colleges and Universities in Shandong Province (KJ2018BAN046).

Data availability statement Data availability depends upon the request of the researchers.

Declarations

Conflict of interests The authors declared that there is no conflict of interest. At the same time, the authors declared that the work described was original research that has not been published previously, and not under consideration for publication elsewhere, in whole or in part.

Informed consent All authors have read this manuscript and are willing to process it for publication.

Ethical approval There is no need for ethical approval while conducting the study in this manuscript.

References

[1] Y. Zhang, Y. Zhou, and S. School, “Review of clustering algorithms,” J. Comput. Appl., 2019.
[2] W. A. Barbakh, W. Ying, and C. Fyfe, “Review of Clustering Algorithms,” Springer Berlin Heidelb., 2009.
[3] E. Bajal, V. Katara, M. Bhatia, and M. Hooda, “A review of clustering algorithms: Comparison of DBSCAN and K-mean with oversampling and t-SNE,” Recent Patents Eng., vol. 15, 2021.
[4] M. Hai, S. Y. Zhang, and M. A. Yan-Lin, “Algorithm review of distributed clustering problem in distributed environments,” Appl. Res. Comput., vol. 30, no. 9, pp. 2561–2564, 2013.
[5] K. Djouzi and K. Beghdad-Bey, “A Review of Clustering Algorithms for Big Data,” in International Conference on Networking and Advanced Systems, 2019.
[6] P. Luo, Q. Huang, and A. Tung, “A Generic Distributed Clustering Framework for Massive Data,” 2021.
[7] E. Januzaj, H. P. Kriegel, and M. Pfeifle, “DBDC: Density Based Distributed Clustering,” DBLP, 2004.
[8] L. I. Liu, “k-DmeansWM:An Effective Distributed Clustering Algorithm Based on P2P,” Comput. Sci., 2010.
[9] M. Ester, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” Proc.int.conf.knowledg Discov. Data Min., 1996.
[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content-Addressable Network,” ACM SIGCOMM Comput. Commun. Rev., vol. 31, no. 4, 2001.
[11] H. C. Ryu and S. Jung, “MapReduce-based Distributed Clustering Method using CF+ tree,” IEEE Access, vol. PP, no. 99, p. 1, 2020.
[12] T. H. Sardar and Z. Ansari, “MapReduce-based Fuzzy C-means Algorithm for Distributed Document Clustering,” 2021.
[13] T. H. Sardar and Z. Ansari, “Distributed Big Data Clustering using MapReduce-based Fuzzy C-Medoids,” J. Inst. Eng. Ser. B, pp. 1–10, 2021.
[14] C. M. Dasari and R. Bhukya, “MapReduce paradigm: DNA sequence clustering based on repeats as features,” Expert Syst., vol. 39, 2022.
[15] Q. Z. Y. L. J. Z. K. Z. Q. W. L. Hu, “Parallel Spectral Clustering Based on MapReduce,” Zte Commun. Technol. English version, no. 2, 2022.
[16] A. E. Abdallah, “A Robust Distributed Clustering of Large Data Sets on a Grid of Commodity Machines,” Data, vol. 6, 2021.
[17] D. Yu, Y. Ying, L. Z Ha Ng, C. Liu, and H. Zheng, “Balanced scheduling of distributed workflow tasks based on clustering,” Knowledge-Based Syst., vol. 199, p. 105930, 2020.
[18] Y. A. Geng, Q. Li, M. Liang, C. Y. Chi, J. Tan, and H. Huang, “Local-Density Subspace Distributed Clustering for High-Dimensional Data,” IEEE Trans. Parallel Distrib. Syst., vol. 31, no. 8, pp. 1799–1814, 2020.
[19] H. E. Tong, X. U. Wei-Hong, M. A. Hong-Hua, and S. L. Zeng, “An Efficient Distributed Clustering Algorithm Based on Peak Density,” Comput. Technol. Autom., 2019.
[20] R. Corizzo, G. Pio, M. Ceci, and D. Malerba, “DENCAST: distributed density-based clustering for multi-target regression,” J. Big Data, vol. 6, no. 1, 2019.
[21] E. Januzaj, H. P. Kriegel, and M. Pfeifle, “Towards Effective and Efficient Distributed Clustering,” Work. Clust. Large Data Sets, 2004.
[22] S. Demirci, A. Yardimci, M. Sayit, E. T. Tunali, and H. Bulut, “A Hierarchical P2P Clustering Framework for Video Streaming Systems,” Comput. Stand. Interfaces, vol. 49, pp. 44–58, 2017.
[23] G. Kai and Z. Liu, “A New Efficient Hierarchical Distributed P2P Clustering Algorithm,” in Fifth International Conference on Fuzzy Systems & Knowledge Discovery, 2008.
[24] L. Yang, C. Zhong, and L. U. Xiang-Yan, “Advances for Distributed Clustering Algorithms Based on P2P Networks,” Microelectron. Comput., vol. 26, no. 8, pp. 83–85, 2009.
[25] H. Mo and S. Guo, “A Distributed Node Clustering Mechanism in P2P Networks,” in Advanced Data Mining and Applications - 6th International Conference, ADMA 2010, Chongqing, China, November 19-21, 2010, Proceedings, Part II, 2010.
[26] M. Li, G. Lee, W. C. Lee, and A. Sivasubramaniam, “PENS: An Algorithm for Density-Based Clustering in Peer-to-Peer Systems,” in International Conference on Scalable Information Systems, 2006.
[27] H. V Jagadish, “BATON: A Balanced Tree Structure for Peer-to-Peer Networks.,” in International Conference on Very Large Data Bases, 2005.
[28] A. Rowstron, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” Ifip/acm Int. Conf. Distrib. Syst. Platforms Open Distrib. Process. Springer, 2003.
[29] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakr-Ishnan, “Chord: A scalable content-addressable network,” Proc Acm Sigcomm, 2001.
[30] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, “MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data,” Front. Comput. Sci., 2014.