Received December 11, 2020, accepted December 19, 2020, date of publication December 24, 2020, date of current version January 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3047094

PSubCLUS: A Parallel Subspace Clustering Algorithm Based On Spark

XIAO WEN[1] AND HU JUAN[2,3]
[1]School of Educational Science, Anhui Normal University, Wuhu 241002, China
[2]Key Laboratory of Unmanned Aerial Vehicle Development and Data Application, Anhui Higher Education Institutes, Wanjiang University of Technology, Maanshan 243000, China
[3]Ma'anshan Engineering Technology Research Center for Wireless Sensor Network and IntelliSense, Wanjiang University of Technology, Maanshan 243000, China

Corresponding author: Xiao Wen (cyees@163.com)

This work was supported by an academic funding project for top discipline talents of the Anhui Provincial Department of Education in 2020 under Grant gxbjzd2020101.

ABSTRACT Clustering is one of the most important unsupervised machine learning tasks. It is widely used to solve problems of intrusion detection, text analysis, image segmentation etc. Subspace clustering is the most important method for high-dimensional data clustering. In order to solve the problem of parallel subspace clustering for high-dimensional big data, this paper proposes a parallel subspace clustering algorithm based on spark named PSubCLUS which is inspired by SubCLU, a classical subspace clustering algorithm. While Spark is the most popular big data parallel processing platform currently, PSubCLUS uses the Resilient Distributed Datasets (RDD) provided by Spark to store data points in a distributed way. The two main performing stages of this algorithm, one-dimensional subspace clustering and iterative clustering, can be executed in parallel on each worker node of cluster. PSubCLUS also uses a repartition method based on the number of data points to achieve load balancing. Experimental results show that PSubCLUS has good parallel speedup and ideal load balancing effect, which is suitable for solving the parallel subspace clustering of high-dimensional big data.

INDEX TERMS Big data applications, clustering algorithms, parallel.

I. INTRODUCTION

Clustering is one of the most important unsupervised machine learning tasks whose purpose is to divide a set of objects into multiple groups or clusters; Objects in the same cluster have high similarity, but are not similar to the objects in other clusters. Clustering comes from Machine Learning, Artificial Intelligence, Data Mining, Bioinformatics, Statistics and other research fields, and is widely used to solve the problems of intrusion detection [1], consumer segmentation [2], text analysis [3], image segmentation [4] etc.

Researchers have proposed many clustering algorithms in recent decades [5]–[9], and these algorithms can be roughly divided into five categories [10]: (1) K-means [11], k-medoids [12] and other algorithms based on partition; (2) BIRCH [13], CURE [14], CHAMELEON [15] and other hierarchical-based algorithms; (3) DBSCAN [16], OPTICS [17], DENCLUE [18] and other density-based algorithms; (4) Grid-based algorithms such as STRING [19], OPTIGRID [20];(5) model-based algorithms such as EM [21], COBWEB [22]. These algorithms mentioned above can meet the needs of clustering small low dimensional datasets.

Datasets generated from bioinformatics, e-commerce, wireless sensor networks, social networks and many other fields generally have many attributes, so they are called high-dimensional data. Traditional clustering algorithms work well when the dimensionality of datasets is not high (datasets have less than 10 attributes), but they are not suitable for high-dimensional data with ten or more attributes. The main reason is that with the increase of the number of attributes in the datasets, the traditional clustering algorithms based on distance or density are unable to calculate the similarity between objects effectively, which makes the algorithms invalid described as "Curse of dimension" [23].

In order to deal with curse of dimension, researchers have proposed a series of clustering algorithms for high-dimensional data [24], which can be divided into two categories. In the first kind of methods, dimension reduction is

The associate editor coordinating the review of this manuscript and approving it for publication was Noor Zaman.

used to reduce the number of attributes by selecting or reducing the original attributes. The typical methods are Principal Component Analysis(PCA) [25] and mRMR [26], etc. This kind of methods selects or regulates the original attributes from the global perspective. Although it can identify and retain the attributes with the highest correlation and avoid the influence of noise attributes on the clustering results, this kind of methods clusters in a subset of the original attributes, ignores the impact of attribute selection on the clustering results, and results in a great loss of information. In addition, there is a serious drawback of this kind of methods: the new attributes generated by dimension reduction are not clear, and the results of clustering are poorly interpretable. The second kind of methods is called subspace clustering. This kind of methods finds clusters from all the subsets of original attributes. The typical algorithms include CLIQUE [27], ENCLUS [28], etc. This kind of methods can find different clusters from different attribute subsets, reveal the local correlation between attributes. The results of clustering are well interpretable, which is the main method for clustering high-dimensional data. SUBCLU [29] is a density-based subspace clustering algorithm, which can find clusters of arbitrary shape in subspace, and the meaning of subspace is intuitive and clear. The algorithm also uses the Apriori property to prune the subspaces that need to be checked, which can greatly reduce the number of subspaces. It is one of the best subspace clustering algorithms for high-dimensional data [30].

With the rapid development of computer, communication, sensor and other technologies, people and equipment are producing massive data at an amazing speed every day. We have entered the era of big data. The traditional centralized clustering algorithms based on a single machine cannot meet the needs of clustering big data. It is necessary to study parallel clustering algorithms to meet the challenges brought by big data. Spark [31] is an open source big data processing framework supported by Apache Software Foundation. It uses memory-based Resilient Distributed Datasets(RDD) as the abstract structure of data. Compared with the traditional Hadoop[32] based on HDFS, Spark has performance advantages of hundreds of times, and is one of the most popular platforms for parallel processing of big data. Due to the characteristics of memory-based execution in Spark, it is particularly suitable for clustering analysis that requires multiple iterations. Researchers have proposed many parallel clustering algorithms based on Spark [33]. We can further improve the performance of subspace clustering algorithms for high-dimensional big data by using Spark.

In this paper, a parallel subspace clustering algorithm named Parallel SubCLU based on Spark (PSubCLUS) is proposed. The algorithm uses Spark RDD to store data in a distributed manner, and uses the functions provided by Spark to implement parallel execution of one-dimensional subspace clustering and iterative clustering which are the two main execution stages of PSubCLUS. The proposed algorithm is suitable for parallel subspace clustering of high-dimensional big data on a cluster. The main contributions of this paper are as follows:

- A parallel subspace clustering algorithm based on Spark named PSubCLUS is proposed. The overall framework of PSubCLUS and the specific design of each execution step are included in the paper.
- A RDD repartition method based on the number of data points is proposed to ensure that PSubCLUS achieves load balancing effectively.
- The parallel acceleration, node scalability and load balancing performance of PSubCLUS are verified by experiments.

The organization of this paper is as follows: In Section II, we introduce the definition and related symbols of subspace clustering problem, and briefly introduce the SubCLU and Apache spark framework. In Section III, the proposed parallel clustering algorithms based on Spark are summarized. The main framework of PSubCLUS and the specific design of each execution step are proposed in Section IV. We analyze the experimental results in Section V. The sixth section summarizes the whole paper. The Conclusion of this paper is in Section VI.

II. RELATED WORKS

Because Spark is especially suitable for clustering that requires multiple iterations, researchers have proposed many parallel clustering algorithms based on Spark. These algorithms use RDD to store the set of data points, use the functions provided by Spark to execute the key steps of classical clustering algorithms in parallel, and achieve wonderful effect.

K-means is the most famous and commonly used partition-based clustering algorithm. There are three key steps: the selection of the initial centroid, the calculation of the distance between each data point and the centroid, and dividing each data point into the cluster which has shortest distance between data point and centroid. Parallel algorithms based on K-means mainly include parallel K-means clustering algorithm in MLlib [35] and parallel k-means algorithm proposed in reference [36], [37]. Wang *et al.* proposed a series of optimization strategies for parallel K-means [38] in MLlib. A parallel intelligent K-means algorithm based on Spark is proposed in [39]. The difference between this algorithm and K-means algorithm is the way of initialization centroid generation. Similar work has been done in [40], which uses bat [41] and firefly [42] algorithms to optimize the selection of initialization centroid. The parallel clustering algorithm proposed by Lu *et al.* [43] adopted tabu search strategy [44], [45] to optimize the centroid update mode. Kacem *et al.* [46] implemented the K-Prototypes [47] based on Spark. The difference between K-Prototypes algorithm and classical K-means algorithm is that the distance between data points and centroid is measured, which can support data points with categorical attributes.

The hierarchical-based clustering algorithms organize all data points into a tree, which can be done by condensing a single data point (bottom-up) or dividing the whole set of data points(bottom-up). The machine learning algorithm library based on Spark named MLlib [35] also contains a hierarchical clustering algorithm; it is a parallel implementation of bisection k-means algorithm [48], which is developed based on paper [49]. Jin *et al.* proposed SHAS [50] that parallelizes the classical SHC algorithm using Spark. The algorithm includes three stages: data point division, local clustering and merging.

The two kinds of algorithms discussed above can only find clusters with spherical shape, and the quality of clustering results is greatly affected by outliers. The density based clustering algorithm can overcome the above two shortcomings. Density based spatial clustering of applications with noise (DBSCAN) [16] is one of the most classic density-based clustering algorithms. It can identify clusters of arbitrary shape and efficiently identify outliers. Fang *et al.* proposed a parallel DBSCAN algorithm based on Spark named parallelDBSCAN [51]. According to the analysis of DBSCAN, more than 90% of its execution time is used to find the core points. Therefore, parallelDBSCAN performs this step in parallel, which significantly improves the efficiency of clustering. Amar *et al.* proposed a parallel clustering algorithm based on shared nearest neighbor (SNN) [52] named SparkSNN [53]. Unlike parallelDBSCAN which uses Euclidean distance as the basis for similarity measurement between data points, SparkSNN uses the number of data points in the intersection of two neighborhoods as the basis of similarity measurement, and data points whose density is greater than the specified threshold(MinPts) are regarded as the core points (centroid of cluster). The overall framework of the REMOLD [54] proposed by Liang *et al.* is consistent with that of parallelDBSCAN, but further optimization methods are adopted in the three stages of data point distribution, local cluster generation and local cluster merging.

The model-based clustering algorithm assumes that the data points come from the data sources containing multiple subpopulations. The data points in each subpopulation conform to a certain probability distribution, and the set of data point is a mixture of multiple subpopulations, the goal of clustering is to divide the data points into several sets that conform to the probability distribution model. The most commonly used probability distribution model is Gaussian mixture models (GMM) [55], which regards the cluster in the set of data points as Gaussian distribution with different parameters, and the whole data point set is a mixture of multiple Gaussian distributions. The process of clustering is to divide each data point into a Gaussian distribution and generate clusters directly. So the model-based clustering algorithm has more advantages than other algorithms in running speed. The machine learning algorithm library based on Spark MLlib [35] contains a parallel clustering algorithm based on GMM. It uses expectation maximization (EM) [56] to find the Gaussian distribution of one or more variables and train the Gaussian mixture model. Through iterative training, the mean and standard deviation of each Gaussian distribution close to the real situation are obtained. The trained Gaussian mixture model is used to classify all data points, and the predefined number of clusters is directly obtained.

TABLE 1. Symbols used in this paper.

Symbol	Quantity
DB	Set of data points
A	Set of attributes in DB
C^S	All clusters in subspace S
S_k	All k-dimensional subspaces containing at least one cluster
|S|	Cardinality of subspace S
$CandS_{k+1}$	Set of candidate k+1-dimensional subspaces
Ck	Set of all clusters in k-dimensional subspace
ε	Radius of DBSCAN algorithm
m	MinPts of DBSCAN algorithm
$s.attr_i$	i-th property of subspace s

III. PROBLEM DEFINITION AND PRELIMINARIES

A. PROBLEM DEFINITION AND SYMBOLS

Let $D = \{o_1, o_2, \ldots, o_n\}$ is a set of objects, $A = \{a_1, a_2, \ldots, a_n\}$ is a set of attributes. Dataset can be expressed as a matrix $DB = D \times A$. A value of attribute a is represented as x_a. we use x_{oa} to express the value of attribute a of object o.

If $O \subseteq D$ is a set of objects and $S \subseteq A$ is a set of attributes, then S is a subspace of DB, and |S| is called cardinality of subspace dimensions. The matrix $C = O \times S$ is called a subspace cluster in DB. All possible subspace clusters in DB can be represented as a set $ALL = \{C_1, C_2, \ldots, C_n\} = \{(O_1 \times S_1), (O_2 \times S_2), \ldots, (O_n \times S_n)\}$. Given that homogeneous function H (C) is used to measure the homogeneity of C, and u is a user-defined homogeneity threshold, then the goal of subspace clustering problem is to find all subspace clusters $M = \{C_1, C_2, \ldots, C_m | h(C_i) \geq u\}$, $M \subseteq ALL$. In a subspace cluster $C = O \times S$, for $\forall o \in O$, o can also belong to other subspace clusters $o \in C'$.

B. SubCLU

SubCLU was proposed by Kailing et al in 2004, and it is one of the best subspace clustering algorithms up to now [10]. Its subspace search direction is bottom-up, which starts from one-dimensional subspace and gradually extends to multi-dimensional subspace, and finds clusters based on density in all subspaces. The main characteristics of SubCLU algorithm are as follows:

- In each subspace, it uses density-based algorithm called DBSCAN for clustering, so it can find clusters with arbitrary shape and position in the subspace.
- Unlike the grid-based algorithms such as CLUQUE, each subspace generated by the SubCLU has a clear meaning.
- For each subspace, the cluster generated by the SubCLU algorithm is stable.

In addition to the above three characteristics, the performance of SubCLU is optimized by using the monotonicity of the density-connected set. On the one hand, according to the monotonicity of the subspace, we do not have to examine any subspace S if at least one $T_i \subset S$ contains no cluster. On the other hand, according to the monotonicity of density-connected data points, there must be $C_{k+1} \subset C_k$ in the process of searching C_{k+1} in k+1-th iteration. Therefore, we only need to check $o \in C_k$ instead of $o \in DB$. The pseudocode of SubCLU is shown in algorithm 1.

Algorithm 1 Pseudocode of SubCLU

```
Input: DB, $\varepsilon$, m
Output: ALL Subspace clusters in DB
1:  $S_1 = \emptyset$
2:  $C_1 = \emptyset$
3:  FOREACH $a_i \in A$
4:      $C^{\{ai\}} = DBSCAN(DB, \{a_i\}, \varepsilon, m)$
5:      IF $C^{\{ai\}} \neq \emptyset$
6:          $S_1 = S_1 \cup \{a_i\}$
7:          $C_1 = C_1 \cup C^{\{ai\}}$
8:      End IF
9:  End For
10: k = 1
11: WHILE $C_k \neq \emptyset$
12:     $CandS_{k+1} = GenerateCandidateSubspaces(S_k)$
13:     FOREACH $cand \in CandS_{k+1}$
14:         $bestSubspace \min_{s \in Sk \wedge S \subseteq cand} = \sum_{C_i \in C^s} |c_i|$
15:         $C^{cand} = \emptyset$
16:         FOREACH $cl \in C^{bestSubspace}$
17:             $C^{cand} = C^{cand} \cup DBSCAN(cl, cand, \varepsilon, m)$
18:             IF $C^{cand} \neq \emptyset$
19:                 $S_{k+1} = S_{k+1} \cup cand$
20:                 $C_{k+1} = C_{k+1} \cup C^{cand}$
21:             END IF
22:         END FOR
23:     END FOR
24:     $S_k = S_{k+1}$
25:     $C_k = C_{k+1}$
26:     k = k + 1
27:
    END WHILE
```

From Algorithm 1, we can see that the SubCLU algorithm is divided into two main stages. In the first stage, all one-dimensional subspaces are clustered to generate S1 and C1 (lines 1-9). The second stage is iterative clustering, which searches for C_{k+1} and S_{k+1} from the discovered C_K and S_K (lines 10-27). In the k-th iteration, the set of k+1-dimensional candidate subspace $CandS_{k+1}$ is generated according to S_k(line 12). In the process of generation, the Apriori property can be used to prune $CandS_{k+1}$ to reduce the size of candidate subspace to be checked. For each subspace cand in $CandS_{k+1}$, the subspace with the least data points in C_k is selected as the best subspace (line 14), and the C^{cand} is found from each subspace cluster of the best subspace (line 17). If the C^{cand} is not empty, it indicates that there are still subspace clusters in the subspace cand. Then, cand is taken as an element in S_{k+1}, and cand is added to C_{k+1}(lines 18-20). The algorithm iterates continuously according to the above process until $S_{k+1} = \emptyset$.

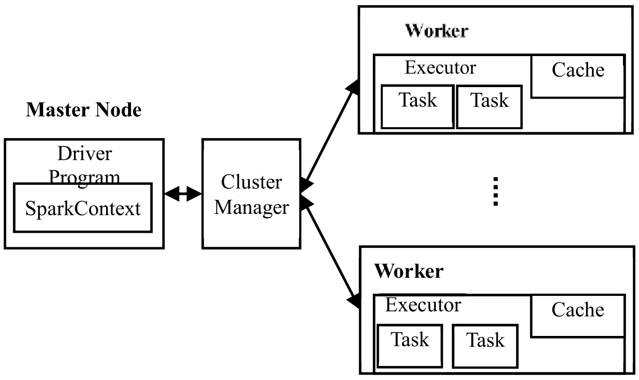


FIGURE 1. Working model of spark framework.

C. APACHE SPARK

Spark [24] is a general distributed computing framework and one of the most popular big data parallel processing platforms. Because it uses a memory-based Resilient Distributed Datasets(RDD) to store input and intermediate results, Spark reduces a lot of I/O cost compared with Hadoop, and is especially suitable for computing tasks requiring multiple iterations [34]. A Spark cluster consists of a master node and many worker nodes. The master node is responsible for managing the resources of work nodes and assigning tasks to worker nodes, while worker nodes execute distributed tasks. The working model of Spark is shown in Figure 1.

Spark can distribute RDD to worker nodes in the cluster automatically or manually. At the same time, Spark provides a number of functions to implement parallel operations on RDD stored across worker nodes. The functions used in this paper are as follows:

- mapToPair(func): Return a new RDD by applying a function to all elements of this RDD.
- flatMapToPair(func): Return a new RDD by first applying a function func to all elements of this RDD, and then flattening the results.
- groupByKey(nums): Group the values for each key in the RDD into a single sequence and repartition.
- partitionBy(Partitioner): Return a copy of the RDD partitioned using the specified partitioner.
- filter(func): Return a new RDD containing only the elements that satisfy a predicate.
- collect (): Return an array that contains all of the elements in this RDD.

IV. PROPOSED PSubCLUS ALGORITHM

A. MAIN STRATEGY OF PSubCLUS

Inspired by SubCLU, PSubCLUS proposed in this paper is a parallel subspace clustering algorithm based on Spark. Its main design strategies are as follows:

- Distributed data storage: the set of data points can be stored in a general distributed file system (such as HDFS). RDD is used to load the set of data points in HDFS to achieve the distributed storage of dataset among cluster nodes.
- The parallel execution of main steps: PSubCLUS mainly includes two execution steps. In the one-dimensional subspace clustering stage, the main execution step is to find subspace clusters in each one-dimensional subspace $\{a_i\}$. In the iterative clustering stage, the main execution step is to find the subspace clusters in each subspace cluster of the best subspace of cand $\in CandS_{k+1}$. All one-dimensional subspaces or cand and their corresponding data points can be distributed to multiple worker nodes (divided into multiple partitions), and the parallel execution on multiple nodes can be realized by using functions provided by Spark.
- Load balancing: Load balancing is one of the most critical factors to achieve good performance for parallel algorithm. Because the space or time complexity of clustering algorithm is directly proportional to the size of the data points, we can achieve the load balance of PSubCLUS by balancing the distribution of data points among nodes.

B. FLOWCHART AND PSEUDO CODE OF PSubCLUS

Similar to SubCLU, PSubCLUS contains two main stages: in the one-dimensional subspace clustering stage, PSubCLUS detects the subspace clusters in each one-dimensional subspace $\{a_i\}\in A$. Since each one-dimensional subspace is independent of each other, this stage can be executed in parallel on multiple nodes. The second stage is iterative clustering stage. In each iteration, the candidate subspace $CandS_{k+1}$ is generated according to the discovered subspace set S_k, and C_{k+1} is detected in the best subspace of each candidate subspace cand$\in CandS_{k+1}$. Because subspace clusters in the best subspace of each cand are independent of each other, the subspace clusters in the best subspace of each cand can be distributed to multiple nodes for parallel execution.

The flowchart of PSubCLUS is shown in Figure 2.

Pseudo code of PSubCLUS is shown in algorithm 2.

From Figure 2 and Algorithm 2, we can see that PSubCLUS sets the task name, cluster and other running conditions firstly, and creates the spark running environment SparkContext (line 1) in the driver program. Then, PSubCLUS reads the set of data points from the distributed file system HDFS, and uses the funcKV to convert the set of data points into Key-Value PairRDD in the form of $<a_i$,List $<$datapoint $\gg$ (line 2). In order to improve the parallelism of the following steps furtherly, this algorithm redistributes rddDB among worker nodes in cluster (line 3), and nums is the number of nodes (partitions). In the one-dimensional subspace clustering stage, PSubCLUS detects subspace clusters in each one-dimensional subspace $\{a_i\}\in A$ (line 4), and uses the func1DClustering to generate Key-Value PairRDD named rddS_1andC_1 in the form of $<$Subspace,List $<$Clusters $\gg$ with mapToPair() provided by Spark. Then, rddS_1andC_1 is redistributed among the worker nodes in the cluster in order to achieve Load Balance (line 5). In each iteration, the candidate subspace $CandS_{k+1}$ is generated according to the discovered subspaces S_k (line 9). The best subspace of each candidate subspace cand$\in$CandSk+1 is selected from C_k (line 10). Then, C_{k+1} is found in the clusters from every best subspace (line 11). The filter () and flatMapToPair () provided by Spark are used to select the best subspace and detect cluster respectively.

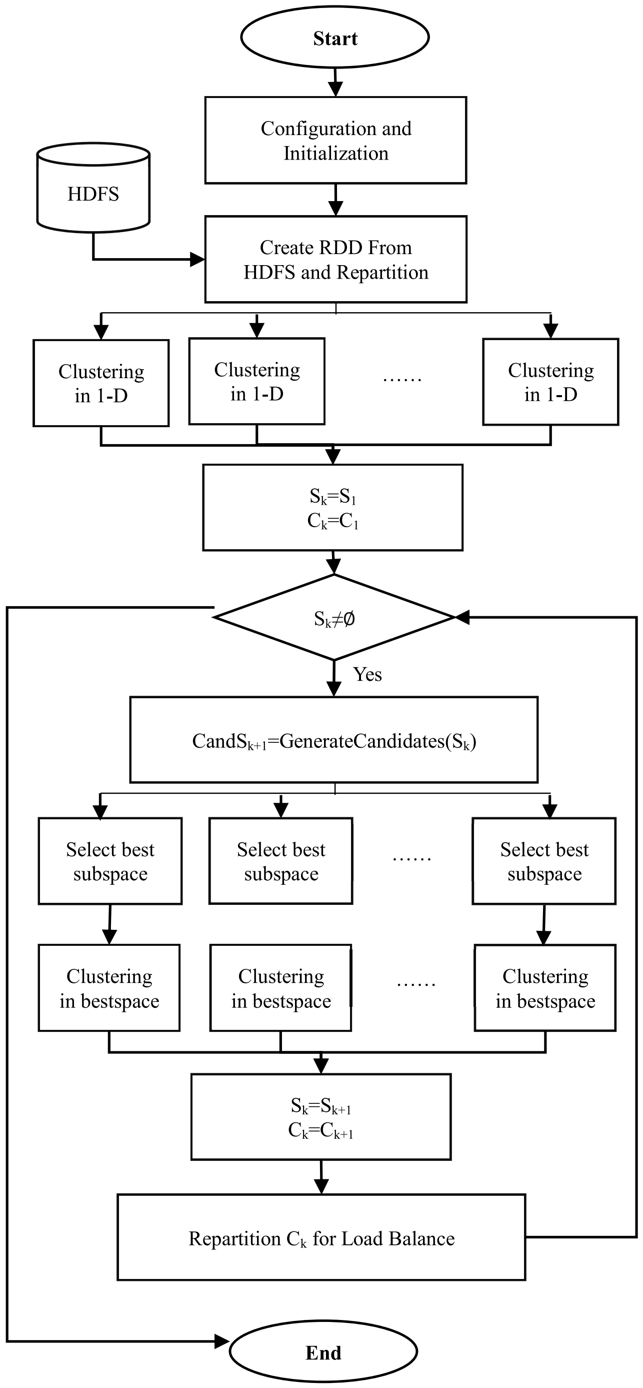


FIGURE 2. Flowchart of PSubCLUS.

Algorithm 2 Pseudocode of PSubCLUS

```
Input: DB, $\varepsilon$, m
Output: ALL Subspace clusters in DB
1:  sc = SparkContext(conf)
2:  rddDB = sc.readFromHDFS(file).flatMapToPair
    (funcKV)
3:  rddDB.repartition(nums)
4:  $rddS_1andC_1$ = rddDB.mapToPair(func1DClustering)
5:  $rddS_1andC_1$.partitionBy(balancePartitioner)
6:  $S_k$ = $rddS_1andC_1$.keyscollect()
7:  $C_k$ = $rddS_1andC_1$
8:  WHILE $S_k \neq \emptyset$
9:      $S_{k+1}$ = GenerateCandidates($S_k$)
10:     rddBest = $C_k$.filter(funcSelectBestSubspace)
11:     $rddS_{k+1}andC_{k+1}$ = rddBest.flatMapToPair
          (funcClustring)
12:     $rddS_{k+1}andC_{k+1}$.partitionBy
        (balancePartitioner)
13:     $S_k$ = $rddS_{k+1}andC_{k+1}$.keyscollect()
14:     $C_k$ = $rddS_{k+1}andC_{k+1}$
15: END WHILE
```

In order to achieve load balancing, the algorithm redistributes $rddS_{k+1}andC_{k+1}$ among worker nodes in cluster (line 12). PSubCLUS ends until S_k is empty (line 8).

C. CLUSTERING IN ONE-DIMENSIONAL SUBSPACE USING func1Dclustering

PSubCLUS algorithm uses func1Dclustering to cluster in one-dimensional subspace, and uses mapToPair() provided by Spark to execute func1Dclustering in parallel among worker nodes in cluster. The pseudo code of func1Dclustering is shown in Algorithm 3.

Algorithm 3 Pseudo Code of func1Dclustering

```
Input: <$a_i$,List<datapoints>> rddDBEle, $\varepsilon$, m
Output: <subspace,List<Cluster>$C_1$Ele
1:  listClusters = DBSCAN(rddDBEle._2, rddDBEle._1,
    $\varepsilon$, m)
2:  IF listClusters $\neq \emptyset$
3:    RETURN < rddDBEle._1, listClusters> $C_1$Ele
```

func1Dclustering uses the classic DBSCAN to find the subspace clusters in the one-dimensional subspace $\{a_i\}$ (line 1), rddDBEle. _1 is $\{a_i\}$, rddDBEle. _2 is List<datapoints> corresponding to $\{a_i\}$. If there is subspace cluster in subspace, the subspace and the set of clusters are regarded as elements in $rddS_1andC_1$.

D. RDD PARTITIONER balancePartitioner

In order to achieve load balancing, PairRDD such as $rddS_1andC_1$ and $rddS_{k+1}andC_{k+1}$ in the form of <subspace, List<Cluster>> should be redistributed among worker nodes in the cluster. The goal of repartition is to make the number of data points processed by each worker node basically equal.

Algorithm 4 Pseudocode of GenerateCanddidates

```
Input: $S_k$
Output: $S_{k+1}$
1:  $CandS_{k+1} = \emptyset$
2:  FOREACH $s_1 \in S_k$ DO
3:    FOREACH $s_2 \in S_k$ DO
4:      IF $s_1.attr_1 = s_2.attr_1$^...^$s_1.attr_{k-1} = s_2.attr_{k-1}$^
        $s_1.attr_k < s_2.attr_k$
5:        Insert $\{s_1.attr_1, \ldots, s_1.attr_k, s_2.attr_k\}$
          into $CandS_{k+1}$
6:      END IF
7:    END FOR
8:  END FOR
9:  FOREACH cand $\in CandS_{k+1}$ DO
10:   FOREACH T $\subset$ cand with |T| = k DO
11:     IF T $\notin S_k$ THEN
12:       delete cand from $CandS_{k+1}$
13:     END IF
14:   ENDFOR
15:
    ENDFOR
```

Based on this goal, balancePartitioner first sums up the total number of data points in List<Cluster> corresponding to each subspace, and then sorts all subspaces according to the total number of data points in descending. Finally, elements in form of <subspace, List<Cluster>> in $rddS_1andC_1$ and $rddS_{k+1}andC_{k+1}$are reallocated to new partition by using Round-Bin.

E. GENERATE S_{k+1} USING GenerateCanddidates

PSubCLUS uses GenerateCanddidates to generate S_{k+1} from S_k which has been discovered, and prunes S_{k+1} by Apriori property, that is, every k-dimensional subspace T in cand((T$\subset$cand)^(|T| = k)^(cand$\in S_{k+1}$)) must exist in S_k, otherwise cand can be pruned from S_{k+1}. The pseudo code of GenerateCanddidates is shown in Algorithm 4.

Algorithm 5 Pseudocode of funcSelectBestSubspace

```
Input: $S_{k+1}$,$C_k$
Output: rddBest
1:  FOREACH s$\in S_{k+1}$ DO
2:    FOREACH sb$\subset$s ^|sb| = k
3:      IF sb$\in C_k$ ^minsum(count(sb.List<Cluster>))
4:        bestSubspace = sb
5:        RETURN <sb, List<Cluster>>
6:      END FOR
7:    END FOR
```

In Algorithm 4, GenerateCanddidates first generates $CandS_{k+1}$ according to S_k (lines 2-8). The method of $CandS_{k+1}$ generation is to join two subspaces in S_k that meet the conditions in line 4. After joining, the elements in $CandS_{k+1}$ are pruned according to Apriori property to get S_{k+1} (lines 9-15).

F. SELECT BEST SUBSPACE USING *funcSelectBestSubspace*

PSubCLUS uses funcSelectBestSubspace to select best subspace of each $s \in S_{k+1}$ from C_k based on the least number of data points in clusters which correspond to the subspace. The pseudo code of this method is described in algorithm 5.

Algorithm 6 Pseudocode of funcClustring

```
Input: <bestSubspace,List<Cluster>rddBestElement
Output: <Subspace, List<Cluster>> $rddS_{k+1}andC_{k+1}Ele$
1: FOREACH cl∈rddBestElement._1 DO
2:     listClusters = DBSCAN(rddBestEle._2, cl, $\varepsilon$, m)
3:     IFlistClusters≠ Ø
4:         RETURN< Subspace, listClusters> Ele
5:     END IF
6: END FOR
```

In Algorithm 5, for each $s \in S_{k+1}$, we first find all k-dimensional subspaces of s in C_k, and select the k-dimensional subspace with the least number of data points in its corresponding subspace cluster as the best subspace of s.

G. CLUSTERING IN SUBSPACE USING *funcClustring*

For rddBest, parallel clustering in each subspace using funcClustring is similar to clustering in one-dimensional subspace. Both of them use classic DBSCAN. The main difference is that funcClustring needs to cluster all subspace clusters corresponding to a best subspace. The pseudo code of this method is described in Algorithm 6.

In algorithm 6, rddBestElement. _2 is a best subspace of $s \in S_{k+1}$; rddBestEle. _1 is the set of all subspace clusters corresponding to the best subspace. The method funcClustring uses DBSCAN to find new subspace clusters in subspace clusters of the best subspace (line 2). If there is at least a new subspace cluster (line 3), subspace s and newly discovered subspace cluster is regarded as an element in $rddS_{k+1}andC_{k+1}$.

V. EXPERIMENTAL RESULTS AND ANALYSIS

In order to verify the performance of the proposed PSubCLUS in terms of speedup, scalability of nodes and load balancing, three groups of experiments are designed as follow:

- The first group of experiments aim to analyze the hotspots of the Non-Parallel version of PSubCLUS and verify the possibility of the performance improvement of the parallel version PSubCLUS.
- The second group of experiments aim to analyze the Parallel Acceleration (or scalability of nodes) of PSubCLUS. We adjust parallelism of PSubCLUS by changing the number of partitions, and observe the change of time spent.
- The third group of experiments aim to verify the effectiveness of the proposed balancePartitioner of RDD, and observe the number of data points processed by each node (or contained in each partition) in different condition of parallelism.

TABLE 2. The basic characteristics of datasets and parameters used by DBSCAN.

NO	Name	Number of Instances	Number of Attributes	m	ε
D1	Anuran Calls	7195	22	300	0.05
D2	Asian Religion Data	590	8265	280	0.5
D3	Gesture Phase Segmentation	9900	50	500	0.1
D4	Sales Transactions Dataset Weekly	811	53	300	2
D5	Tarvel Review Ratings	5456	25	85	0.1
D6	Mice Protein Expression	1080	82	125	0.05

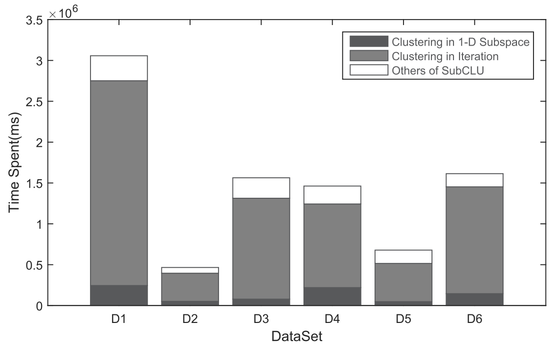


FIGURE 3. Hotspots of non-parallel version of PSubCLUS.

TABLE 3. The number of data points in each partition when number of partitions is 4.

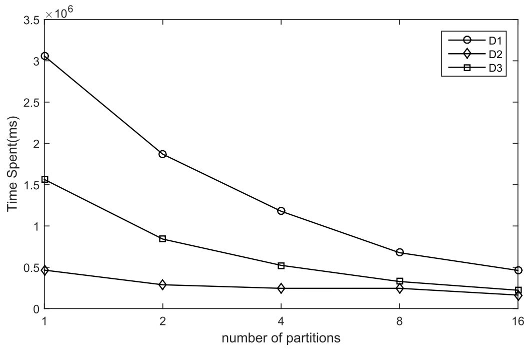
NO of Partition	The number of data points in each partition					
	D1	**D2**	**D3**	**D4**	**D5**	**D6**
0	362467	19423	350382	43953	70632	95796
1	358076	19464	350987	43638	71338	94719
2	355784	19151	338948	43363	70806	94300
3	357483	19257	340577	43397	72284	95093

The datasets used in experiments are all from UC machine learning repository [57]. The basic attributes of the datasets and the parameters used by DBSCAN are shown in Table 2.

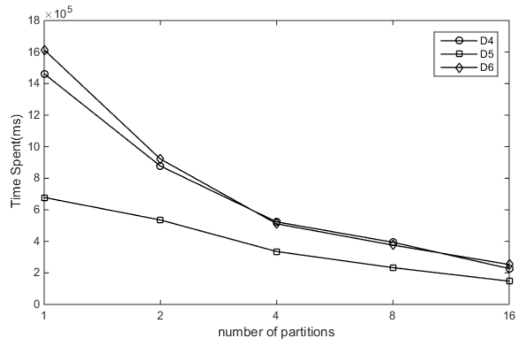
Experiments use Dell PowerEdge T430 server as the test machine, and its CPU is Intel Xeon e5-2620 V4@2.10GHz (8 cores, 16 threads), size of RAM is 64GB, OS is Windows Server 2008. PSubCLUS proposed in this paper is programming in Java. The version of JDK is jdk-8u191-windows-x64, and the version of Spark is 2.47.

A. HOTSPOTS OF NON-PARALLEL VERSION OF PSubCLUS

PSubCLUS proposed in this paper includes two main execution stages: one-dimensional subspace clustering and clustering in iteration. If the main execution stages of the



A) DATASET D1-D3



B) DATASET D4-D6

FIGURE 4. Experimental results of parallel speedup of PSubCLUS.

algorithm are able to execute in parallel, the performance of the algorithm can be obviously improved. In this group of experiments, we use Non-Parallel version of PSubCLUS to clustering datasets in Table 2, and verify the hotspots of this algorithm. The experimental results are shown in Figure 3.

From Figure 3, we can see that there is an obvious hotspot of Non-Parallel version of PSubCLUS, that is, clustering in iteration. This stage accounts for about 80% of all the time spent by the algorithm. Therefore, parallel execution of this stage can significantly improve the performance of PSubCLUS. From Figure 3, we can also see that although the one-dimensional subspace clustering stage accounts for about 10% of all the time spent by the algorithm, the parallel execution of this stage is very easy to achieve. Therefore, PSubCLUS proposed in this paper will also execute the one-dimensional subspace clustering stage in parallel, which further improves the efficiency.

B. PARALLEL SPEEDUP

In this group of experiments, we adjust the parallelism of PSubCLUS by setting different numbers of partitions of RDD to verify the parallel speedup of the proposed algorithm. When the number of partitions of RDD is one, PSubCLS becomes a Non-Parallel version. The DBSCAN algorithm used in PSubCLUS uses the values of m and ε in Table 2 respectively. The experiment uses different parallelism of PSubCLUS to cluster the datasets in Table 2, and the time spent is shown in Figure 4.

TABLE 4. The number of data points in each partition when number of partitions is 8.

NO of Partition	The number of data points in each partition					
	D1	D2	D3	D4	D5	D6
0	180314	9636	179713	21551	41514	48423
1	180049	9655	180015	21565	41658	47670
2	181139	9682	168614	21582	41230	46962
3	182217	9728	169699	21561	41238	47311
4	182206	9771	170296	21250	41229	47056
5	178090	9794	171265	20944	41823	46630
6	174704	9820	172481	20959	42370	46986
7	175355	9507	173370	20632	41964	47366

TABLE 5. The number of data points in each partition when number of partitions is 16.

NO of Partition	The number of data points in each partition					
	D1	D2	D3	D4	D5	D6
0	92051	4928	92538	10921	21296	24833
1	93280	4936	93033	10949	24264	23853
2	93888	4957	83149	10955	21459	24000
3	94545	4991	83645	10934	21514	23407
4	94991	5020	84450	10609	20579	23595
5	90882	5034	84978	10613	21062	23748
6	87265	5053	85584	10624	20784	23952
7	87599	4698	85116	10627	21134	24204
8	88257	4708	87310	10621	19304	23811
9	86766	4719	86973	10288	19373	23998
10	87250	4725	85535	10294	19424	23157
11	87671	4737	85179	10298	19487	23304
12	87212	4751	84659	10298	19868	23474
13	87194	4760	84958	10308	19964	23585
14	87426	4767	85593	10311	20187	23773
15	87739	4809	86575	10316	20375	23988

From Figure 4, we can see that with the continuous increase of parallelism (the number of partitions), the time spent by PSubCLUS on the six datasets is reduced correspondingly, and the time spent on the algorithm is basically inversely proportional to the parallelism. This shows that the algorithm has good parallel speedup, and it will have better performance when executed on a cluster with more worker nodes.

C. LOAD BALANCING

Load balancing is one of the most important factors affecting the performance of parallel algorithms, which requires that tasks undertaken by each worker node are basically equal. PSubCLUS uses the RDD partitioner named balancepartitioner described in part D in Section IV to repartition $\mathrm{rddS_1andC_1}$ and $\mathrm{rddS_{k+1}andC_{k+1}}$ to ensure that each partition (worker node) processes the almost same number of data points. In this group of experiments, the total number of data points in each partition under different parallelism (number of partitions) was counted. When the number of partitions is 4, 8 and 16, the total number of data points in each partition is shown in table 3-5.

From Table 3-5, we can see that balancePartitioner proposed in part D in Section IV achieves very good load balancing effect. Under the condition of different parallelism (the number of partitions), the number of data points in each partition is basically the same, which enables PSubCLUS to execute in parallel on each worker node in the cluster evenly, showing good performance.

VI. CONCLUSION AND FUTURE WORK

In order to solve the problem of high-dimensional big data clustering, this paper has proposed a parallel subspace clustering algorithm based on Spark named PSubCLUS. Inspired by the classic subspace clustering algorithm named SubCLU, PSubCLUS implements parallel execution of one-dimensional subspace clustering and iterative clustering using the functions provided by Spark. In order to achieve the effect of load balancing, this paper has proposed a method of RDD repartition based on the number of data points, which ensures that the number of data points in each partition is almost equal, so that each worker nodes can perform balanced when PSubCLUS is running on a cluster which contains many worker nodes.

In the future, we will carry out further research on handling complex data and improving clustering results. Firstly,in addition to the traditional 2D dataset, people or machines in many fields also produce complex high-dimensional dataset such as 3D data with timestamp, categorical data, stream data and so on, which brings new challenges to high-dimensional data clustering. Secondly,subspace clustering needs to find clusters in each subspace. Although subspaces need to be checked can be pruned by monotonicity, the number of them is still huge. We need to find special subspaces based on entropy or interesting in all subspaces, which are called significant subspace, to further improving clustering results.

REFERENCES

[1] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Syst. Appl.*, vol. 36, no. 10, pp. 11994–12000, 2009.

[2] H.-H. Liu and C.-S. Ong, "Variable selection in clustering for marketing segmentation using genetic algorithms," *Expert Syst. Appl.*, vol. 34, no. 1, pp. 502–510, Jan. 2008.

[3] C.-E. Ben N'Cir and N. Essoussi, "Using sequences of words for non-disjoint grouping of documents," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 29, no. 03, May 2015, Art. no. 1550013.

[4] H. Zhang, J. E. Fritts, and S. A. Goldman, "Image segmentation evaluation: A survey of unsupervised methods," *Comput. Vis. Image Understand.*, vol. 110, no. 2, pp. 260–280, May 2008.

[5] A. K. Jain, M. Narasimha, M. Murty, M. Patrick, and J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.

[6] Berkhin, Pavel, "A survey of clustering data mining techniques," in *Grouping Multidimensional Data*. Berlin, Germany: Springer, 2006, pp. 25–71.

[7] J. Grabmeier and A. Rudolph, "Techniques of cluster algorithms in data mining," *Data Mining Knowl. Discovery* vol. 6, no. 4, pp. 303–360, 2002.

[8] R. Xu and D. Wunsch, II, "Survey of clustering algorithms," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 645–678, May 2005.

[9] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, Jun. 2010.

[10] D. Pandove, S. Goel, and R. Rani, "Systematic review of clustering high-dimensional and large datasets," *ACM Trans. Knowl. Discovery Data*, vol. 12, no. 2, pp. 1–68, Mar. 2018.

[11] MacQueen, James, "Some methods for classification and analysis of multivariate observations," *Proc. Berkeley Symp. Math. Statist. Probability*, 1967, vol. 1, no. 14, p. 15.

[12] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for K-medoids clustering," *Expert Syst. Appl.*, vol. 36, no. 2, pp. 3336–3341, Mar. 2009.

[13] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," *ACM SIGMOD Rec.*, vol. 25, pp. 103–114, Jun. 1996.

[14] S. Guha, R. Rastogi, and K. Shim, "CURE: An efficient clustering algorithm for large databases," *ACM SIGMOD Rec.*, vol. 27, no. 2, pp. 73–84, 1998.

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

[16] A. Ram, S. Jalal, A. S. Jalal, and M. Kumar, "A density based algorithm for discovering density varied clusters in large spatial databases," *Int. J. Comput. Appl.*, vol. 3, no. 6, pp. 1–4, Jun. 2010.

[17] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," *ACM SIGMOD. Rec.*, vol. 28, no. 2, pp. 49–60, Jun. 1999.

[18] A. Hinneburg and D. A. Keim, "An efficient approach to clustering in large multimedia databases with noise," in *Proc. AAAI*, 1998, pp. 58–65.

[19] W. Wang, J. Yang, and R. Muntz, "STING: A statistical information grid approach to spatial data mining," in *Proc. VLDB*, vol. 97, 1997, pp. 186–195.

[20] A. Hinneburg and D. A. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering," in *Proc. 25th VLDB Conf.*, 1999.

[21] A. P. Dempster and M. B. Nan Laird Donald Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc. B, Methodol.*, vol. 39, no. 1, pp. 1–22, 1977.

[22] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Mach. Learn.*, vol. 2, no. 2, pp. 139–172, Sep. 1987.

[23] F. Y. Kuo and I. H. Sloan, "Lifting the curse of dimensionality," *Notices AMS*, vol. 52, no. 11, pp. 1320–1328, 2005.

[24] H.-P. Kriegel, P. Kröger, and A. Zimek, "Detecting clusters in moderate-to-high dimensional data: Subspace clustering, pattern-based clustering, and correlation clustering," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1528–1529, Aug. 2008.

[25] Wold, Svante, Kim Esbensen, and Paul Geladi, "Principal component analysis," *Chemometrics Intell. Lab. Syst.*, vol. 2, nos. 1–3, pp. 37–52, 1987.

[26] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1226–1238, Aug. 2005.

[27] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1998, pp. 94–105.

[28] C.-H. Cheng, A. W. Fu, and Y. Zhang, "Entropy-based subspace clustering for mining numerical data," in *Proc. 5th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 1999, pp. 84–93.

[29] K. Kailing, H.-P. Kriegel, and P. Kröger, "Density-connected subspace clustering for high-dimensional data," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2004, pp. 246–256.
[30] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: A review," *ACM SIGKDD Explor. Newslett.*, vol. 6, no. 1, pp. 90–105, Jun. 2004.
[31] M. Zaharia *et al.*, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.
[32] V. K. Vavilapalli, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, and H. Shah, "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, pp. 1–17.
[33] W. Xiao and J. Hu, "A survey of parallel clustering algorithms based on spark," *Sci. Program.*, vol. 2020, pp. 1–12, Sep. 2020.
[34] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, Berkeley, CA, USA, Jul. 2010, pp. 1–5.
[35] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska, "MLI: An API for distributed machine learning," in *Proc. IEEE 13th Int. Conf. Data Mining*, Dec. 2013, pp. 1187–1192.
[36] B. Shangguan and P. Yue, "SPARK processing of computing-intensive classification of remote sensing images: The case on K-means clustering algorithm," in *Proc. 26th Int. Conf. Geoinformat.*, Jun. 2018, pp. 1–4.
[37] X. Mallios, V. Vassalos, T. Venetis, and A. Vlachou, "A framework for clustering and classification of big data using spark," in Proceedings of the OTM Confederated," in *Proc. Int. Conf. Move Meaningful Internet Syst.* Rhodes, Greece: Springer, Oct. 2016, pp. 344–362.
[38] B. Wang, J. Yin, Q. Hua, and Z. Wu, "Parallelizing k-meansbased clustering on spark," in *Proc. Int. Conf. Adv. Cloud Big Data (CBD)*, Chengdu, China, Aug. 2016, pp. 1–5.
[39] I. Kusuma, M. A. Ma'sum, N. Habibie, W. Jatmiko, and H. Suhartanto, "Design of intelligent k-means based on spark for big data clustering," in *Proc. Int. Workshop Big Data Inf. Secur. (IWBIS)*, Oct. 2016, pp. 1–5.
[40] V. Santhi and R. Jose, "Performance analysis of parallel K-means with optimization algorithms for clustering on spark," in *Proc. Int. Conf. Distrib. Comput. Internet Technol.* Bhubaneswar, India: Springer, Jan. 2018, pp. 1–8.
[41] X.-S. Yang, "Bat algorithm: Literature review and applications," 2013, *arXiv:1308.3900*. [Online]. Available: http://arxiv.org/abs/1308.3900
[42] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Prc. Int. Symp. Stochastic Algorithms* Sapporo, Japan: Springer, Oct. 2009, pp. 1–6.
[43] Y. Lu, B. Cao, C. Rego, and F. Glover, "A tabu search based clustering algorithm and its parallel implementation on spark," *Appl. Soft Comput.*, vol. 63, pp. 97–109, Feb. 2018.
[44] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.
[45] F. Glover, "Tabu Search—Part II," *ORSA J. Comput.*, vol. 2, no. 1, pp. 4–32, Feb. 1990.
[46] M. A. Ben HajKacem, C. E. Ben N'Cir, and N. Essoussi, "KP-S: A spark-based design of the K-Prototypes clustering for big data," in *Proc. IEEE/ACS 14th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Oct. 2017, pp. 557–563.
[47] Z. Huang, "Clustering large data sets with mixed numeric and categorical values," in *Proc. 1st Pacific-Asia Conf. Knowl. Discouvery Data Mining*, River Edge, NJ, USA, Feb. 1997, pp. 1–5
[48] S. M. Savaresi and D. L. Boley, "On the performance of bisecting K-means and PDDP," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2001, pp. 1–4.
[49] S. Hong, W. Choi, and W.-K. Jeong, "GPU in-memory processing using spark for iterative computation," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017.
[50] C. Jin, "A scalable hierarchical clustering algorithm using spark," in *Proc. IEEE 1st Int. Conf. Big Data Comput. Service Appl.*, Redwood City, CA, USA, Mar. 2015, pp. 418–426.
[51] F. Huang, Q. Zhu, J. Zhou, J. Tao, X. Zhou, D. Jin, X. Tan, and L. Wang, "Research on the parallelization of the DBSCAN clustering algorithm for spatial data mining based on the spark platform," *Remote Sens.*, vol. 9, no. 12, p. 1301, Dec. 2017.
[52] L. Ertöz, M. Steinbach, and V. Kumar, "Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data," in *Proc. SIAM Int. Conf. Data Mining*, May 2003, pp. 47–58.
[53] A. M. Aryal and S. Wang, "SparkSNN: A density-based clustering algorithm on spark," in *Proc. IEEE 3rd Int. Conf. Big Data Anal. (ICBDA)*, Mar. 2018, pp. 433–437.
[54] M. Liang, Q. Li, Y.-A. Geng, J. Wang, and Z. Wei, "REMOLD: An efficient model-based clustering algorithm for large datasets with spark," in *Proc. IEEE 23rd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2017, pp. 376–383.
[55] R. P. Browne, P. D. McNicholas, and M. D. Sparling, "Modelbased learning using a mixture of mixtures of Gaussian and uniform distributions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 4, pp. 814–817, Dec. 2011.
[56] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EMAlgorithm," *J. Roy. Stat. Soc., B Methodol.*, vol. 39, no. 1, pp. 1–22, 1977.
[57] A. Asuncion and D. Newman, "UCI machine learning repository," Center Mach. Learn. Intell. Syst., 2007.

XIAO WEN received the Ph.D. degree from Hohai University, China, in 2020. He is currently with Anhui Normal University. His research interests include big data mining and distributed computing. He is a member of the China Computer Federation.

HU JUAN received the master's degree from Nanjing Normal University, China, in 2010. She has been with the Wanjiang University of Technology since 2010. Her research interests include big data mining and machine learning.